

# BACKBONE.JS

# 目錄

# Backbone.js(1.1.2) API中文文档

> 来源：Backbone.js(1.1.2) API中文文档




## 文档声明：

- 根据官方最新的1.1.2版本做了翻译
- Backbone.js (0.5.3)文档请移步http://www.css88.com/doc/backbone-0.5.3/；
- 翻译水平有限，如果您有任何建议，或者拍砖，欢迎在微博上@愚人码头 联系我。

Backbone.js为复杂WEB应用程序提供模型(**models**)、集合(**collections**)、视图(**views**)的结构。其中模型用于绑定键值数据和自定义事件；集合附有可枚举函数的丰富API； 视图可以声明事件处理函数，并通过RESRful JSON接口连接到应用程序。

此项目托管在 GitHub 上, 并且提供 带注释的源码, 在线的 测试套件, 应用示例, 教程列表 还有一个 实际应用项目的超长列表, 这些项目都使用了 Backbone. Backbone 允许在 MIT 软件协议 下使用.

你可以在GitHub issues 页面及 `#documentcloud` 频道下的Freenode IRC中 报告 bug 和 讨论功能, 在Google Group合wiki页面中提出问题，或在twitter上 @documentcloud。

*Backbone是 DocumentCloud 的一个开源组件.*

## 下载&依赖 (右键另存为)

| | |
|---|---|
| 开发版本 (1.1.2) | *60kb,* 完整的源代码，大量的注释 |
| 生产版本 (1.1.2) | *6.5kb,* 打包和*gzip*压缩 (Source Map) |
| Edge Version (master) | 未发布版本,使用风险自负 build failing |

Backbone唯一重度依赖的是Underscore.js( >= 1.5.0)（愚人码头注：Underscore.js 中文文档请查看 http://www.css88.com/doc/underscore/）。基于RESTful（一个架构样式的网络系统）的约束，histroy的支持依赖于Backbone.Router ，DOM处理依赖于 Backbone.View，包括**jQuery** ( >= 1.11.0), 和 **json2.js**对旧的IE浏览器的支持。（模仿*Underscore* 和 *jQuery* 的 *APIs*，例如 *Lo-Dash* 和 *Zepto*，在不同的兼容性下也一样能运行）

# 介绍（Introduction）

当我们开发含有大量Javascript的web应用程序时，首先你需要做的事情之一便是停止向DOM对象附加数据。 通过复杂多变的jQuery选择符和回调函数很容易创建Javascript应用程序，包括在HTML UI，Javascript逻辑和数据之间保持同步，都不复杂。 但对富客户端应用来说，良好的架构通常是有很多益处的。

通过Backbone，你可以将数据呈现为 Models, 你可以对模型进行创建，验证和销毁，以及将它保存到服务器。 任何时候只要UI事件引起模型内的属性变化，模型会触发*"change"*事件； 所有显示模型数据的 Views 会接收到该事件的通知，继而视图重新渲染。 你无需查找DOM来搜索指定*id*的元素去手动更新HTML。 — 当模型改变了，视图便会自动变化。

某种意义上说，在用javaScript来创建web项目时，Backbone试图定义一组最小而高效的集合，包括了数据结构（models（模型） 和 collections（集合））和用户接口（views（视图） 和 URLS）。在web开发环境里，到处都是框架（帮你写好了一切），不过这些库需要你的网站在构建的时候符合该框架的样子，风格，默认的行为。但是，Backbone还是作为一个工具，让你可以随心所欲的设计你的网站。

如果你不懂Backbone或者不确定Backbone能否帮助到你，先运行一下 列表中基于Backbone的项目。

文档下面有大量可以运行的例子，请点击 *play* 执行他们。

# 升级到 1.1

从Backbone **0.9.X**系列版本升级到 **1.1** 应该是相当容易的。如果你从旧版本升级，一定要检查更新日志。简单地说，一些大规模的重大更改是：

- 如果你想漂亮的更新一个Collection(集合)的内容，增加新的models（模型），删除丢失，和合并那些已经存在，你现在可以调用set(以前叫做"update")，Model（模型）类似的操作也调用 `set` 。这是目前默认的，当你在collection（集合）上调用fetch时。为了得到旧的行为，传递 `{reset: true}` 。
- 如果你的URL片段中有字符，需要URL编码，Backbone现在会在你路由处理程序接收它们作为参数前为你解码（跨浏览器规范的行为）。
- 在**0.9.x**中，Backbone 事件有了两个新的方法：listenTo 和 stopListening， 这使得它能更容易地创建Views（视图）监听，当你想 remove view（视图）时，解除他们所有绑定的监听。
- model（模型）验证现在只默认执行在save中 —不再执行在set中，除非传递了 `{validate:true}` 选项。model（模型）验证现在会触发一个 `"invalid"` 事件，而不是 `"error"` 事件。
- 在1.1中 ，Backbone Views（视图）不再有 `options` 参数自动附加在 `this.options` 上。如果你喜欢可以继续附加。
- 在1.1中 ，**Collection**的 `add` , `remove` , `set` , `push` , 和 `shift` 方法现在返回来自collection（集合）的 models（模型）(或 models)。

# Backbone.Events（事件）

**Events** 是一个可以融合到任何对象的模块, 给予 对象绑定和触发自定义事件的能力. Events 在绑定之前 不需要声明, 并且还可以传递参数. 比如:

```
var object = {};

_.extend(object, Backbone.Events);

object.on("alert", function(msg) {
  alert("Triggered " + msg);
});

object.trigger("alert", "an event");
```

举个例子, 你可以定义一个事件调度程序，然后在你应用的不同地方调用，例如： `var dispatcher = _.clone(Backbone.Events)`

**on** `object.on(event, callback, [context])` 别名: bind 在 object 上绑定一个**callback**回调函数。 只要**event**触发，该回调函数就会调用。如果你的一个页面含有大量的不同时间，我们约定使用冒号来为事件添加 命名空间 俗成地使用冒号来命名：`"poll:start"`, 或 `"change:selection"` 。 事件字符串也可能是用空格分隔的多个事件列表（愚人码头注：即可以同时绑定多个事件，事件用空格分隔）...

```
book.on("change:title change:author", ...);
```

当回调函数被调用时，通过可选的第三个参数可以为 `this` 提供一个**context**（上下文）值： `model.on('change', this.render, this)` （愚人码头注：即回调函数中的This，指向传递的第三个参数）。

当回调函数被绑定到特殊"all"事件时，任何事件的发生都会触发该回调函数，回调函数的第一个参数会传递该事件的名称。举个例子，将一个对象的所有事件代理到另一对象：

```
proxy.on("all", function(eventName) {
  object.trigger(eventName);
});
```

所有Backbone事件方法还支持事件映射的语法， 作为可惜的位置参数：

```
book.on({
  "change:title": titleView.update,
  "change:author": authorPane.update,
  "destroy": bookView.remove
});
```

**off** `object.off([event], [callback], [context])` 别名: unbi nd 从 object 对象移除先前绑定的 **callback** 函数。如果没有指定**context**（上下文），所有上下文下的这个回调函数将被移除。如果没有指定callback，所有绑定这个事件回调函数将被移除；如果没有指定event，所有事件的回调函数会被移除。

```
// Removes just the `onChange` callback.
object.off("change", onChange);

// Removes all "change" callbacks.
object.off("change");

// Removes the `onChange` callback for all events.
object.off(null, onChange);

// Removes all callbacks for `context` for all events.
object.off(null, null, context);

// Removes all callbacks on `object`.
object.off();
```

需要注意的是，调用 `model.off()` ，例如，这确实会删除model（模型）上所有的事件—包括Backbone内部用来统计的事件。

**trigger** `object.trigger(event, [*args])` 触发给定 **event**或用空格隔开的事件的回调函数。后续传入 **trigger** 的参数会传递到触发事件的回调函数里。

**once** `object.once(event, callback, [context])` 用法跟on很像，区别在于绑定的回调函数触发一次后就会被移除（愚人码头注：只执行一次）。简单的说就是“下次不在触发了，用这个方法”。

**listenTo** `object.listenTo(other, event, callback)` 让 **object** 监听 另一个（**other**）对象上的一个特定事件。不使用 `other.on(event, callback, object)` ，而使用这种形式的优点是：**listenTo**允许 **object**来跟踪这个特定事件，并且以后可以一次性全部移除它们。**callback**总是在**object**上下文环境中被调用。

```
view.listenTo(model, 'change', view.render);
```

**stopListening** `object.stopListening([other], [event], [callback])` 让 **object** 停止监听事件。如果调用不带参数的**stopListening**，可以移除 **object** 下所有已经registered(注册)的callback函数...，或者只删除指定对象上明确告知的监听事件，或者一个删除指定事件，或者只删除指定的回调函数。

```
view.stopListening();

view.stopListening(model);
```

**listenToOnce** `object.listenToOnce(other, event, callback)` 用法跟 listenTo 很像，但是事件触发一次后callback将被移除。

**Catalog of Events**（事件目录） 下面是Backbone 内置事件的完整列表，带有参数。 你也可以在Models（模型），Collection（集合），Views（视图）上自由地触发这些事件，只要你认为合适。 收藏和意见，你认为合适。 `Backbone` 对象本身混入了 `Events` ，并且可用于触发任何全局事件，只要您的应用程序的需要。

- **"add"** (model, collection, options) — 当一个model（模型）被添加到一个collection（集合）时触发。
- **"remove"** (model, collection, options) — 当一个model（模型）从一个collection（集合）中被删除时触发。
- **"reset"** (collection, options) — 当该collection（集合）的全部内容已被替换时触发。
- **"sort"** (collection, options) — 当该collection（集合）已被重新排序时触发。
- **"change"** (model, options) — 当一个model（模型）的属性改变时触发。
- **"change:[attribute]"** (model, value, options) — 当一个model（模型）的某个特定属性被更新时触发。
- **"destroy"** (model, collection, options) —当一个model（模型）被destroyed（销毁）时触发。
- **"request"** (model_or_collection, xhr, options) — 当一个model（模型）或collection（集合）开始发送请求到服务器时触发。
- **"sync"** (model_or_collection, resp, options) — 当一个model（模型）或collection（集合）成功同步到服务器时触发。
- **"error"** (model_or_collection, resp, options) — 当一个model（模型）或collection（集合）的请求远程服务器失败时触发。
- **"invalid"** (model, error, options) — 当model（模型）在客户端 validation（验证）失败时触发。
- **"route:[name]"** (params) — 当一个特定route（路由）相匹配时通过路由器触发。
- **"route"** (route, params) — 当_任何一个_route（路由）相匹配时通过路由器触发。
- **"route"** (router, route, params) — 当_任何一个_route（路由）相匹配时通过history（历史记录）触发。
- **"all"** — 所有事件发生都能触发这个特别的事件，第一个参数是触发事件的名称。

一般来说，事件触发（例如model.set，collection.add或者其他事件）后就会执行回调函数，但是如果你想阻止回调函数的执行，你可以传递{silent: true}作为参数。很多时候，这是一个好的方法。通过在回调函数里传输一个特定的判断参数，会让你的程序更加出色。 一般而言，事件触发（ `model.set` , `collection.add` ,等等...）后就会调用一个函数，但是如果你想阻止事件被触发， 您可以传递 `{silent: true}` 作为一个选项。注意，这中情况很少， 甚至从来没有， 一个好主意。 通过在选项中传递一个特定的标记，回调函数里传输一个特定的判断参数 并且选择忽略，会让你的程序更加出色。

# Backbone.Model（模型）

**Models**（模型）是任何Javascript应用的核心，包括数据交互及与其相关的大量逻辑： 转换、验证、计算属性和访问控制。你可以用特定的方法扩展 **Backbone.Model**，**Model** 也提供了一组基本的管理变化的功能。

下面的示例演示了如何定义一个模型，包括自定义方法、设置属性、以及触发该属性变化的事件。一旦运行此代码后， `sidebar` 在浏览器的控制台就可用，这样你就可以充分发挥了。

```
var Sidebar = Backbone.Model.extend({
  promptColor: function() {
    var cssColor = prompt("Please enter a CSS color:");
    this.set({color: cssColor});
  }
});

window.sidebar = new Sidebar;

sidebar.on('change:color', function(model, color) {
  $('#sidebar').css({background: color});
});

sidebar.set({color: 'white'});

sidebar.promptColor();
```

**extend** `Backbone.Model.extend(properties, [classProperties])` 要创建自己的 **Model** 类，你可以扩展 **Backbone.Model** 并提供实例 **properties(**属性**)** ， 以及可选的可以直接注册到构造函数的**classProperties(**类属性**)**。

**extend** 可以正确的设置原型链，因此通过 **extend** 创建的子类 (subclasses) 也可以被深度扩展。

```
var Note = Backbone.Model.extend({

  initialize: function() { ... },

  author: function() { ... },

  coordinates: function() { ... },

  allowedToEdit: function(account) {
    return true;
  }

});

var PrivateNote = Note.extend({

  allowedToEdit: function(account) {
    return account.owns(this);
  }

});
```

父类（ `super` ） 的简述：Javascript没有提供一种直接调用父类的方式 — 如果你要重载原型链中上层定义的同名函数，如 `set` , 或 `save` ， 并且你想调用父对象的实现，这时需要明确的调用它，类似这样：

```
var Note = Backbone.Model.extend({
  set: function(attributes, options) {
    Backbone.Model.prototype.set.apply(this, arguments);
    ...
  }
});
```

**constructor / initialize** `new Model([attributes], [options])` 当创建model实例时，可以传入属性 (**attributes**)初始值，这些值会被 set （设置）到 model。 如果定义了 **initialize** 函数，该函数会在model创建后执行。

```
new Book({
  title: "One Thousand and One Nights",
  author: "Scheherazade"
});
```

在极少数的情况下，你可能需要去重写 **constructor** ，它可以让你替换你的model的实际构造函数。

```
var Library = Backbone.Model.extend({
  constructor: function() {
    this.books = new Books();
    Backbone.Model.apply(this, arguments);
  },
  parse: function(data, options) {
    this.books.reset(data.books);
    return data.library;
  }
});
```

如果你传入 `{collection: ...}` ，这个 **options**表示这个model属于哪个 `collection` ，且用于计算这个model的url。否则 `model.collection` 这个属性会在你第一次添加model到一个collection的时候被自动添加。 需要注意的是相反的是不正确的，因为传递这个选项给构造函数将不会自动添加model到集合。有时这个是很有用的。

如果 `{parse: true}` 被作为一个**option**选项传递， **attributes**将在 set到model之前首先通过parse被转换。

**get** `model.get(attribute)` 从当前model中获取当前属性(attributes)值，比如： `note.get("title")`

**set** `model.set(attributes, [options])` 向model设置一个或多个hash属性(attributes)。如果任何一个属性改变了model的状态，在不传入 {silent: true} 选项参数的情况下，会触发 `"change"` 事件，更改特定属性的事件也会触发。 可以绑定事件到某个属性，例如： `change:title` ，及 `change:content` 。

```
note.set({title: "March 20", content: "In his eyes she eclipses..."});

book.set("title", "A Scandal in Bohemia");
```

**escape** `model.escape(attribute)` 与 get 类似，只是返回的是HTML转义后版本的model属性值。如果从model插入数据到HTML，使用 escape 取数据可以避免 XSS 攻击。

```
var hacker = new Backbone.Model({
  name: "<script>alert('xss')</script>"
});

alert(hacker.escape('name'));
```

**has** `model.has(attribute)` 属性值为非 null 或非 undefined 时返回 `true` 。

```
if (note.has("title")) {
  ...
}
```

**unset** `model.unset(attribute, [options])` 从内部属性散列表中删除指定属性(attribute)。 如果未设置 `silent` 选项，会触发 `"change"` 事件。

**clear** `model.clear([options])` 从model中删除所有属性， 包括 `id` 属性。 如果未设置 `silent` 选项，会触发 `"change"` 事件。

**id** `model.id` id是model的特殊属性，可以是任意字符串（整型 **id** 或 UUID）。在属性中设置的 **id** 会被直接拷贝到model属性上。 我们可以从集合（collections）中通过 id 获取model，另外 id 通常用于生成model的 URLs。

**idAttribute** `model.idAttribute` 一个model的唯一标示符，被储存在 `id` 属性下。如果使用一个不同的唯一的key直接和后端通信。可以设置Model的 `idAttribute` 到一个从key到 `id` 的一个透明映射中。

```
var Meal = Backbone.Model.extend({
  idAttribute: "_id"
});

var cake = new Meal({ _id: 1, name: "Cake" });
alert("Cake id: " + cake.id);
```

**cid** `model.cid` model的特殊属性，**cid** 或客户 id 是当所有model创建时自动产生的唯一标识符。 客户 ids 在model尚未保存到服务器之前便存在，此时model可能仍不具有最终的 **id**，但已经需要在用户界面可见。

**attributes** `model.attributes` **attributes** 属性是包含模型状态的内部散列表 — 通常（但不一定）JSON对象的形式表示在服务器上模型数据 。它通常是数据库中一个行的简单的序列，但它也可以是客户端的计算状态。

建议采用 set 更新 **attributes**而不要直接修改。 如果您想检索和获取模型属性的副本， 用 `_.clone(model.attributes)` 取而代之。

由于这样的事实： Events（事件）接受空格分隔事件列表， 但是属性名称不应该包括空格。

**changed** `model.changed` **changed**属性是一个包含所有属性的内部散列，自最后 set 已改变。 自最后一组已改变。 请不要直接更新 **changed**，因为它的状态是由set内部维护。 **changed**的副本可从changedAttributes获得。

**defaults** `model.defaults or model.defaults()` **defaults** 散列（或函数）用于为模型指定默认属性。 创建模型实例时，任何未指定的属性会被设置为其默认值。

```
var Meal = Backbone.Model.extend({
  defaults: {
    "appetizer":  "caesar salad",
    "entree":     "ravioli",
    "dessert":    "cheesecake"
  }
});

alert("Dessert will be " + (new Meal).get('dessert'));
```

需要提醒的是，在 Javascript 中，对象是按引用传值的，因此如果包含一个对象作为默认值，它会被所有实例共享。可以定义 **defaults**为一个函数取代。

**toJSON** `model.toJSON([options])` 返回一个模型的 attributes 浅拷贝副本的 JSON 字符串化形式。 它可用于模型的持久化、序列化，或者发送到服务之前的扩充。 该方法名称比较混乱，因为它事实上并不返回 JSON 字符串， 但这是对 JavaScript API 的 **JSON.stringify** 实现。

```
var artist = new Backbone.Model({
  firstName: "Wassily",
  lastName: "Kandinsky"
});

artist.set({birthday: "December 16, 1866"});

alert(JSON.stringify(artist));
```

**sync** `model.sync(method, model, [options])` 使用 Backbone.sync 可以将一个模型的状态持续发送到服务器。 可以自定义行为覆盖。

**fetch** `model.fetch([options])` 通过委托给Backbone.sync从服务器重置模型的状态。返回 jqXHR。 如果模型从未填充数据时非常有用， 或者如果你想确保你有最新的服务器状态。 如果服务器的状态不同于当前属性的 `"change"` 事件将被触发。 接受 `success` 和 `error` 回调的选项散列， 这两个回调都可以传递 `(model, response, options)` 作为参数。

```
// 每隔 10 秒从服务器拉取数据以保持频道模型是最新的
setInterval(function() {
  channel.fetch();
}, 10000);
```

**save** `model.save([attributes], [options])` 通过委托给Backbone.sync，保存模型到数据库（或替代持久化层）。 如果验证成功，返回jqXHR，否则为 `false` 。 **attributes**散列（如set）应包含你想改变的属性 - 不涉及的键不会被修改 - 但是，该资源的一个完整表示将被发送到服务器。 至于 `set` ，你可能会传递单独的键和值，而不是一个哈希值。 如果模型有一个validate方法，并且验证失败， 该模型将不会被保存。 如果模型isNew， 保存将采用 `"create"` （HTTP `POST` ）， 如果模型在服务器上已经存在， 保存将采用 `"update"` （HTTP `PUT` ）。

相反，如果你只想将改变属性发送到服务器， 调用 `model.save(attrs, {patch: true})` 。 你会得到一个HTTP `PATCH` 请求将刚刚传入的属性发送到服务器。

通过新的属性调用 `save` 将立即触发一个 `"change"` 事件，一个 `"request"` 事件作为Ajax请求开始到服务器， 并且当服务器确认成功修改后立即触发 一个 `"sync"` 事件。 如果你想在模型上等待服务器设置新的属性，请传递 `{wait: true}` 。

在下面的例子中， 注意我们如何覆盖 `Backbone.sync` 的版本，在模型初次保存时接收到 `"create"` 请求，第二次接收到 `"update"` 请求的。

```
Backbone.sync = function(method, model) {
  alert(method + ": " + JSON.stringify(model));
  model.set('id', 1);
};

var book = new Backbone.Model({
  title: "The Rough Riders",
  author: "Theodore Roosevelt"
});

book.save();

book.save({author: "Teddy"});
```

**save** 支持在选项散列表中传入 `success` 和 `error` 回调函数， 回调函数支持传入 `(model, response, options)` 作为参数。 如果服务端验证失败，返回非 `200` 的 HTTP 响应码，将产生文本或 JSON 的错误内容。

```
book.save("author", "F.D.R.", {error: function(){ ... }});
```

**destroy** `model.destroy([options])` 通过委托给Backbone.sync，保存模型到数据库（或替代持久化层）。 通过委托一个HTTP `DELETE` 请求给Backbone.sync破坏服务器上的模型。 返回一个jqXHR对象， 或者如果模型isNew，那么返回 `false` 。 选项散列表中接受 `success` 和 `error` 回调函数， 回调函数支持传入 `(model, response, options)` 作为参数。 在模型上触

发 `"destroy"` 事件，该事件将会冒泡到任何包含这个模型的集合中， 一个 `"request"` 事件作为Ajax请求开始到服务器， 并且当服务器确认模型被删除后立即触发 一个 `"sync"` 事件。如果你想在集合中删除这个模型前等待服务器相应，请传递 `{wait: true}` 。

```
book.destroy({success: function(model, response) {
  ...
}});
```

**Underscore** 方法 **(6)** Backbone 代理了 **Underscore.js**用来给**Backbone.Model**提供 6 个对象函数。这里没有完全记录他们，但你可以看看Underscore文档中全部详情… (愚人码头注：下面链接已经替换成中文文档的地址)

- keys
- values
- pairs
- invert
- pick
- omit

```
user.pick('first_name', 'last_name', 'email');

chapters.keys().join(', ');
```

**validate** `model.validate(attributes, options)` 这种方法是未定义的， 如果您有任何可以在JavaScript中执行的代码 并且我们鼓励你用你自定义验证逻辑覆盖它 。 默认情况下**validate**在 `save` 之前调用， 但如果传递了 `{validate:true}` ，也可以在 `set` 之前调用。 **validate**方法是通过模型的属性， 选项和 `set` 和 `save` 是一样的。 如果属性是有效的， **validate**不返回验证任何东西; 如果它们是无效的， 返回一个你选择的错误。 它可以是一个用来显示的简单的字符串错误信息， 或一个以编程方式描述错误的完整错误对象。 如果**validate**返回一个错误， `save` 不会继续， 并且在服务器上该模型的属性将不被修改。 校验失败将触发 `"invalid"` 事件， 并用此方法返回的值设置模型上的 `validationError` 属性。

```
var Chapter = Backbone.Model.extend({
  validate: function(attrs, options) {
    if (attrs.end < attrs.start) {
      return "can't end before it starts";
    }
  }
});

var one = new Chapter({
  title : "Chapter One: The Beginning"
});

one.on("invalid", function(model, error) {
  alert(model.get("title") + " " + error);
});

one.save({
  start: 15,
  end:   10
});
```

`"invalid"` 事件提供粗粒度的错误信息 在模型或集合层面上是很有用。

**validationError** `model.validationError` 用validate最后验证失败时返回的值。

**isValid** `model.isValid()` 运行validate来检查模型状态。

```
var Chapter = Backbone.Model.extend({
  validate: function(attrs, options) {
    if (attrs.end < attrs.start) {
      return "can't end before it starts";
    }
  }
});

var one = new Chapter({
  title : "Chapter One: The Beginning"
});

one.set({
  start: 15,
  end:   10
});

if (!one.isValid()) {
  alert(one.get("title") + " " + one.validationError);
}
```

**url** `model.url()` 返回模型资源在服务器上位置的相对 URL 。 如果模型放在其它地方，可通过合理的逻辑重载该方法。 生成 URLs 的默认形式为： `"/[collection.url]/[id]"` ， 如果模型不是集合的一部分，你可以通过指定明确的 `urlRoot` 覆盖。

由于是委托到 Collection#url 来生成 URL， 所以首先需要确认它是否定义过，或者所有模型共享一个通用根 URL 时，是否存在 urlRoot 属性。 例如，一个 id 为 `101` 的模型，存储在 `url` 为 `"/documents/7/notes"` 的 Backbone.Collection 中， 那么该模型的 URL 为： `"/documents/7/notes/101"`

**urlRoot** `model.urlRoot or model.urlRoot()` 如果使用的集合外部的模型，通过指定 `urlRoot` 来设置生成基于模型 id 的 URLs 的默认 url 函数。 `"[urlRoot]/id"` 。通常情况下，你不会需要定义这一点。 需要注意的是 `urlRoot` 也可以是一个函数。

```
var Book = Backbone.Model.extend({urlRoot : '/books'});

var solaris = new Book({id: "1083-lem-solaris"});

alert(solaris.url());
```

**parse** `model.parse(response, options)` **parse** 会在通过 fetch 从服务器返回模型数据，以及 save 时执行。 传入本函数的为原始 `response` 对象，并且应当返回可以 set 到模型的属性散列表。 默认实现是自动进行的，仅简单传入 JSON 响应。 如果需要使用已存在的 API，或者更好的命名空间响应，可以重载它。

如果使用的3.1版本之前的 Rails 后端，需要注意 Rails's 默认的 `to_json` 实现已经包含了命名空间之下的模型属性。 对于无缝的后端集成环境禁用这种行为：

```
ActiveRecord::Base.include_root_in_json = false
```

**clone** `model.clone()` 返回该模型的具有相同属性的新实例。

**isNew** `model.isNew()` 模型是否已经保存到服务器。 如果模型尚无 `id` ，则被视为新的。

**hasChanged** `model.hasChanged([attribute])` 标识模型从上次 set 事件发生后是否改变过。 如果传入 **attribute** ，当指定属性改变后返回 `true` 。

注意，本方法以及接下来 change 相关的方法，仅对 `"change"` 事件发生有效。

```
book.on("change", function() {
  if (book.hasChanged("title")) {
    ...
  }
});
```

**changedAttributes** `model.changedAttributes([attributes])` 只从最后一次set开始检索已改变的模型属性散列（hash）， 或者如果没有，返回 `false` 。 作为可选，可以传递外部属性哈希（hash）， 返回与该模型不同的属性的哈希（hash）。 这可以用来找出视图的哪些部分应该更新，或者确定哪些需要与服务器进行同步。

**previous** `model.previous(attribute)` 在 `"change"` 事件发生的过程中，本方法可被用于获取已改变属性的旧值。

```
var bill = new Backbone.Model({
  name: "Bill Smith"
});

bill.on("change:name", function(model, name) {
  alert("Changed name from " + bill.previous("name") + " to " + name);
});

bill.set({name : "Bill Jones"});
```

**previousAttributes** `model.previousAttributes()` 返回模型的上一个属性的副本。一般用于获取模型的不同版本之间的区别，或者当发生错误时回滚模型状态。

# Backbone.Collection（集合）

集合是模型的有序组合，我们可以在集合上绑定 `"change"` 事件，从而当集合中的模型发生变化时 `fetch` （获得）通知，集合也可以监听 `"add"` 和 `"remove"` 事件， 从服务器更新，并能使用 Underscore.js 提供的方法。

集合中的模型触发的任何事件都可以在集合身上直接触发，所以我们可以监听集合中模型的变化： `documents.on("change:selected", ...)`

**extend** `Backbone.Collection.extend(properties, [classProperties])` 通过扩展 **Backbone.Collection** 创建一个 **Collection** 类。实例属性参数 **properties** 以及 类属性参数 **classProperties** 会被直接注册到集合的构造函数。

**model** `collection.model` 覆盖此属性来指定集合中包含的模型类。可以传入原始属性对象（和数组）来 add, create,和 reset，传入的属性会被自动转换为适合的模型类型。

```
var Library = Backbone.Collection.extend({
  model: Book
});
```

集合也可以包含多态模型，通过用构造函数重写这个属性，返回一个模型。

```
var Library = Backbone.Collection.extend({

  model: function(attrs, options) {
    if (condition) {
      return new PublicDocument(attrs, options);
    } else {
      return new PrivateDocument(attrs, options);
    }
  }

});
```

**constructor / initialize** `new Backbone.Collection([models], [options])` 当创建集合时，你可以选择传入初始的 **models** 数组。 集合的 comparator 函数也可以作为选项传入。 传递 `false` 作为comparator选项将阻止排序。 如果定义了 **initialize** 函数，会在集合创建时被调用。 有几个选项， 如果提供的话，将直接附加到集合上： `model` 和 `comparator` 。 通过传递 `null` 给 `models` 选项来创建一个空的集合。

```
var tabs = new TabSet([tab1, tab2, tab3]);
var spaces = new Backbone.Collection([], {
  model: Space
});
```

**models** `collection.models` 访问集合中模型的内置的JavaScript 数组。通常我们使用 `get` ， `at` ，或 **Underscore**方法 访问模型对象，但偶尔也需要直接访问。

**toJSON** `collection.toJSON([options])` 返回集合中包含的每个模型(通过 toJSON) 的属性哈希的数组。可用于集合的序列化和持久化。本方法名称容易引起混淆，因为它与 JavaScript's JSON API 命名相同。

```
var collection = new Backbone.Collection([
  {name: "Tim", age: 5},
  {name: "Ida", age: 26},
  {name: "Rob", age: 55}
]);

alert(JSON.stringify(collection));
```

**sync** `collection.sync(method, collection, [options])` 使用 Backbone.sync来将一个集合的状态持久化到服务器。 可以自定义行为覆盖。

**Underscore** 方法 **(32)** Backbone 代理了 **Underscore.js**用来给**Backbone.Collection**提供 6 个对象函数。这里没有完全记录他们，但你可以看看Underscore文档中全部详情…(愚人码头注：下面链接已经替换成中文文档的地址)

- forEach (each)
- map (collect)
- reduce (foldl, inject)
- reduceRight (foldr)
- find (detect)
- filter (select)
- reject
- every (all)
- some (any)
- contains (include)
- invoke
- max
- min
- sortBy
- groupBy
- shuffle
- toArray
- size
- first (head, take)
- initial
- rest (tail, drop)
- last
- without
- indexOf
- lastIndexOf

- isEmpty
- chain
- difference
- sample
- partition
- countBy
- indexBy

```
books.each(function(book) {
  book.publish();
});

var titles = books.map(function(book) {
  return book.get("title");
});

var publishedBooks = books.filter(function(book) {
  return book.get("published") === true;
});

var alphabetical = books.sortBy(function(book) {
  return book.author.get("name").toLowerCase();
});
```

**add** `collection.add(models, [options])` 向集合中增加一个模型（或一个模型数组），触发 `"add"` 事件。 如果已经定义了model属性， 您也可以通过原始属性的对象让其看起来像一个模型实例。 返回已经添加的（或预先存在的，如果重复）模式。 传递 `{at: index}` 可以将模型插入集合中特定的 `index` 索引位置。 如果您要添加 集合中已经存在的模型 到集合，他们会被忽略， 除非你传递 `{merge: true}` ， 在这种情况下，它们的属性将被合并到相应的模型中， 触发任何适当的 `"change"` 事件。

```
var ships = new Backbone.Collection;

ships.on("add", function(ship) {
  alert("Ahoy " + ship.get("name") + "!");
});

ships.add([
  {name: "Flying Dutchman"},
  {name: "Black Pearl"}
]);
```

请注意，添加相同的模型（具有相同 `id` 的模型）到一个集合，一次以上 是空操作。

**remove** `collection.remove(models, [options])` 从集合中删除一个模型（或一个模型数组），并且返回他们。会触发 `"remove"` 事件，同样可以使用 `silent` 关闭。移除前该模型的index可用作 `options.index` 类监听。

**reset** `collection.reset([models], [options])` 每次都是只添加和删除一个模型那没问题， 但有时，你需要改变很多模型，那么你宁愿只更新集合。 使用**reset**，将一个新的模型（或属性散列）列表替换集合，最后触发一个但单独的 `"reset"` 事件。 到与模型的新列表替换集合

（或属性散列），触发一个单一的“复位”事件在末端。 返回新的模型集合。 为方便起见， 在一个 `"reset"` 事件中， 任何以前的模型列表可作为 `options.previousModels` 。 通过传递 `null` 给 `models` 选项来清空你的集合。

下面是一个例子 使用**reset**来引导一个集合在页面初始化时加载， 在Rails应用程序中：

```
<script>
  var accounts = new Backbone.Collection;
  accounts.reset(<%= @accounts.to_json %>);
</script>
```

调用 `collection.reset()` ，不传递任何模型作为参数 将清空整个集合。

**set** `collection.set(models, [options])` **set**方法通过传递模型列表执行一个集合的"smart(智能)"的更新。 如果列表中的一个模型尚不在集合中，那么它将被添加; 如果模型已经在集合中，其属性将被合并; 并且如果集合包含不存在于列表中的任何模型，他们将被删除。 以上所有将触发相应的 `"add"` , `"remove"` , 和 `"change"` 事件。 返回集合中的模型。 如果您想自定义的行为， 你可以设置选项： `{add: false}` , `{remove: false}` , 或 `{merge: false}` ，将其禁用。

```
var vanHalen = new Backbone.Collection([eddie, alex, stone, roth]);

vanHalen.set([eddie, alex, stone, hagar]);

// Fires a "remove" event for roth, and an "add" event for "hagar".
// Updates any of stone, alex, and eddie's attributes that may have
// changed over the years.
```

**get** `collection.get(id)` 通过一个id，一个cid，或者传递一个**model**来 获得集合中 的模型。

```
var book = library.get(110);
```

**at** `collection.at(index)` 获得集合中指定索引的模型。不论你是否对模型进行了重新排序，**at** 始终返回其在集合中插入时的索引值。

**push** `collection.push(model, [options])` 在集合尾部添加一个模型。选项和add相同。

**pop** `collection.pop([options])` 删除并且返回集合中最后一个模型。选项和remove相同。

**unshift** `collection.unshift(model, [options])` 在集合开始的地方添加一个模型。选项和add相同。

**shift** `collection.shift([options])` 删除并且返回集合中第一个模型。选项和remove相同。

**slice** `collection.slice(begin, end)` 返回一个集合的模型的浅拷贝副本，使用与原生Array#slice相同的选项。

**length** `collection.length` 与数组类似，集合拥有 `length` 属性，返回该集合包含的模型数量。

**comparator** `collection.comparator` 默认情况下，集合没有声明 **comparator** 函数。如果定义了该函数，集合中的模型会按照指定的算法进行排序。 换言之，被增加模型，会被插入 `collection.models` 中适合的位置。 comparator可以被定义为sortBy（传递带有一个参数的函数），作为一个sort（传递一个一个参数函数需要两个参数），或者作为一个表示属性的字符串进行排序。

"sortBy"比较函数接受一个模型，并且返回一个该模型相对于其他模型的排序数字或字符串值。 "sort"比较函数接受两个模型，并且，如果第一个模型应该在第二模型个之前，返回 `-1`；如果他们是同一等级的，返回 `0`；如果第一个模型应该在第二模型个之后，返回 `1`；需要注意的是 *Backbone* 这两种风格的比较功能的确定 取决于参数个数。所以如果你绑定了比较函数，需要格外小心。

注意即使下面例子中的 chapters 是后加入到集合中的，但它们都会遵循正确的排序：

```
var Chapter  = Backbone.Model;
var chapters = new Backbone.Collection;

chapters.comparator = 'page';

chapters.add(new Chapter({page: 9, title: "The End"}));
chapters.add(new Chapter({page: 5, title: "The Middle"}));
chapters.add(new Chapter({page: 1, title: "The Beginning"}));

alert(chapters.pluck('title'));
```

如果以后更改模型属性，带有比较函数的集合不会自动重新排序。 所以你不妨改变模型的属性后调用 `sort`， 这会影响排序。

**sort** `collection.sort([options])` 强制对集合进行重排序。一般情况下不需要调用本函数，因为当一个模型被添加时， comparator 函数会实时排序。要禁用添加模型时的排序，可以传递 `{sort: false}` 给 `add`。 调用**sort**会触发的集合的 `"sort"` 事件。

**pluck** `collection.pluck(attribute)` 从集合中的每个模型中拉取 attribute（属性）。等价于调用 `map`，并从迭代器中返回单个属性。

```
var stooges = new Backbone.Collection([
  {name: "Curly"},
  {name: "Larry"},
  {name: "Moe"}
]);

var names = stooges.pluck("name");

alert(JSON.stringify(names));
```

**where** `collection.where(attributes)` 返回集合中所有匹配所传递 **attributes**（属性）的模型数组。 对于简单的 `filter（过滤）` 比较有用。

```
var friends = new Backbone.Collection([
  {name: "Athos",      job: "Musketeer"},
  {name: "Porthos",    job: "Musketeer"},
  {name: "Aramis",     job: "Musketeer"},
  {name: "d'Artagnan", job: "Guard"},
]);

var musketeers = friends.where({job: "Musketeer"});

alert(musketeers.length);
```

**findWhere** `collection.findWhere(attributes)` 就像where， 不同的是**findWhere**直接返回匹配所传递 **attributes**（属性）的第一个模型。

**url** `collection.url or collection.url()` 设置 **url** 属性（或函数）以指定集合对应的服务器位置。集合内的模型使用 **url** 构造自身的 URLs。

```
var Notes = Backbone.Collection.extend({
  url: '/notes'
});

// Or, something more sophisticated:

var Notes = Backbone.Collection.extend({
  url: function() {
    return this.document.url() + '/notes';
  }
});
```

**parse** `collection.parse(response, options)` 每一次调用 fetch 从服务器拉取集合的模型数据时，**parse**都会被调用。 本函数接收原始 `response` 对象，返回可以 added（添加） 到集合的模型属性数组。 默认实现是无需操作的，只需简单传入服务端返回的JSON对象。 如果需要处理遗留API，或者在返回数据定义自己的命名空间，可以重写本函数。

```
var Tweets = Backbone.Collection.extend({
  // The Twitter Search API returns tweets under "results".
  parse: function(response) {
    return response.results;
  }
});
```

**clone** `collection.clone()` 返回一个模型列表完全相同的集合新实例。

**fetch** `collection.fetch([options])` 从服务器拉取集合的默认模型设置 ，成功接收数据后会setting（设置） 集合。 **options** 支持 `success` 和 `error` 回调函数，两个回调函数接收 `(collection, response, options)` 作为参数。当模型数据从服务器返回时， 它使用 set来（智能的）合并所获取到的模型， 除非你传递了 `{reset: true}` ， 在这种情况下，集合将（有效地）重置。 可以委托Backbone.sync 在幕后自定义持久性策略 并返回一个jqXHR。 **fetch**请求的服务器处理器应该返回模型JSON数组。

```
Backbone.sync = function(method, model) {
  alert(method + ": " + model.url);
};

var accounts = new Backbone.Collection;
accounts.url = '/accounts';

accounts.fetch();
```

**fetch**行为可以通过使用有效的set选项进行定制。 例如，要获取一个集合，每一个新的模型会得到一个 `"add"` 事件，和每改变现有的模型的 `"change"` 事件， 不删除任何东西： `collection.fetch({remove: false})`

**jQuery.ajax**选项也可以直接传递作为 **fetch**选项， 所以要获取一个分页集合的特定页面使用： `Documents.fetch({data: {page: 3}})` 。

需要注意的是 **fetch** 不应该被用来在页面加载完毕时填充集合数据 — 所有页面初始数据应当在 bootstrapped 时已经就绪。 **fetch** 适用于惰性加载不需立刻展现的模型数据：例如：例如文档中 可切换打开和关闭的选项卡内容。

**create** `collection.create(attributes, [options])` 方便的在集合中创建一个模型的新实例。 相当于使用属性哈希（键值对象）实例化一个模型， 然后将该模型保存到服务器， 创建成功后将模型添加到集合中。 返回这个新模型。 如果客户端验证失败，该模型将不会被保存， 与验证错误。 为了能正常运行，需要在集合中设置 model 属性。 create 方法接收键值对象或者已经存在尚未保存的模型对象作为参数。

创建一个模型将立即触发集合上的 `"add"` 事件， 一个 `"request"` 的事件作为新的模型被发送到服务器， 还有一个 `"sync"` "事件，一旦服务器响应成功创建模型。 如果你想在集合中添加这个模型前等待服务器相应，请传递 `{wait: true}` 。

```
var Library = Backbone.Collection.extend({
  model: Book
});

var nypl = new Library;

var othello = nypl.create({
  title: "Othello",
  author: "William Shakespeare"
});
```

# Backbone.Router（路由）

web应用程序通常需要为应用的重要位置提供可链接，可收藏，可分享的 URLs。 直到最近，猫点（hash）片段（`#page`）可以被用来提供这种链接， 同时随着 History API 的到来，猫点已经可以用于处理标准 URLs （`/page`）。 **Backbone.Router** 为客户端路由提供了许多方法，并能连接到指定的动作（actions）和事件（events）。 对于不支持 History API 的旧浏览器，路由提供了优雅的回调函数并可以透明的进行 URL 片段的转换。

页面加载期间，当应用已经创建了所有的路由，需要调用 `Backbone.history.start()`，或 `Backbone.history.start({pushState: true})` 来确保驱动初始化 URL 的路由。

**extend** `Backbone.Router.extend(properties, [classProperties])` 开始创建一个自定义的路由类。当匹配了 URL 片段便执行定义的动作，并可以通过 routes 定义路由动作键值对。 请注意，你要避免在路由定义时使用前导斜杠：

```
var Workspace = Backbone.Router.extend({

  routes: {
    "help":                 "help",    // #help
    "search/:query":        "search",  // #search/kiwis
    "search/:query/p:page": "search"   // #search/kiwis/p7
  },

  help: function() {
    ...
  },

  search: function(query, page) {
    ...
  }

});
```

**routes** `router.routes` routes 将带参数的 URLs 映射到路由实例的方法上（或只是直接的函数定义，如果你喜欢），这与 View（视图） 的 events hash（事件键值对） 非常类似。 路由可以包含参数， `:param`，它在斜线之间匹配 URL 组件。 路由也支持通配符， `*splat`，可以匹配多个 URL 组件。 路由的可选部分放在括号中 `(/:optional)`。

举个例子，路由 `"search/:query/p:page"` 能匹配 `#search/obama/p2` , 这里传入了 `"obama"` 和 `"2"` 到路由对应的动作中去了。

路由 `"file/*path"` 可以匹配 `#file/nested/folder/file.txt`，这时传入动作的参数为 `"nested/folder/file.txt"`。

路由 `"docs/:section(/:subsection)"` 可以匹配 `#docs/faq` 和 `#docs/faq/installing`，第一种情况，传入 `"faq"` 到路由对应的动作中去， 第二种情况，传入 `"faq"` 和 `"installing"` 到路由对应的动作中去。

结尾的斜杠会被当作URL的一部分， 访问时会被（正确地）当作一个独立的路由。 `docs` 和 `docs/` 将触发不同的回调。 如果你不能避免产生这两种类型的URLs时， 你可以定义一个 `"docs(/)"` 来匹配捕捉这两种情况。

当访问者点击浏览器后退按钮，或者输入 URL ，如果匹配一个路由，此时会触发一个基于动作名称的 event， 其它对象可以监听这个路由并接收到通知。 下面的示例中，用户访问 `#help/uploading` 将从路由中触发 `route:help` 事件。

```
routes: {
  "help/:page":         "help",
  "download/*path":     "download",
  "folder/:name":       "openFolder",
  "folder/:name-:mode": "openFolder"
}
```

```
router.on("route:help", function(page) {
  ...
});
```

**constructor / initialize** `new Router([options])` 当创建一个新路由是，你可以直接传入 routes 键值对象作为参数。 如果定义该参数， 它们将被传入 `initialize` 构造函数中初始化。

**route** `router.route(route, name, [callback])` 为路由对象手动创建路由， `route` 参数可以是 routing string（路由字符串） 或 正则表达式。 每个捕捉到的被传入的路由或正则表达式，都将作为参数传入回调函数（callback）。 一旦路由匹配， `name` 参数会触发 `"route:name"` 事件。如果 `callback` 参数省略 `router[name]` 将被用来代替。 后来添加的路由可以覆盖先前声明的路由。

```
initialize: function(options) {

  // Matches #page/10, passing "10"
  this.route("page/:number", "page", function(number){ ... });

  // Matches /117-a/b/c/open, passing "117-a/b/c" to this.open
  this.route(/^(.*?)\/open$/, "open");

},

open: function(id) { ... }
```

**navigate** `router.navigate(fragment, [options])` 每当你达到你的应用的一个点时，你想保存为一个URL， 可以调用**navigate**以更新的URL。 如果您也想调用路由功能， 设置**trigger**选项设置为 `true` 。 无需在浏览器的历史记录创建条目来更新URL， 设置 **replace**选项设置为 `true` 。

```
openPage: function(pageNumber) {
  this.document.pages.at(pageNumber).open();
  this.navigate("page/" + pageNumber);
}

# Or ...

app.navigate("help/troubleshooting", {trigger: true});

# Or ...

app.navigate("help/troubleshooting", {trigger: true, replace: true});
```

**execute** `router.execute(callback, args)` 这种方法在路由内部被调用， 每当路由和其相应的**callback**匹配时被执行。 覆盖它来执行自定义解析或包装路由， 例如， 在传递他们给你的路由回调之前解析查询字符串，像这样：

```
var Router = Backbone.Router.extend({
  execute: function(callback, args) {
    args.push(parseQueryString(args.pop()));
    if (callback) callback.apply(this, args);
  }
});
```

# Backbone.history

**History** 作为全局路由服务用于处理 `hashchange` 事件或 `pushState` ，匹配适合的路由，并触发回调函数。 我们不需要自己去做这些事情 — 如果使用带有键值对的 路由， `Backbone.history` 会被自动创建。

Backbone 会自动判断浏览器对 **pushState** 的支持，以做内部的选择。 不支持 `pushState` 的浏览器将会继续使用基于猫点的 URL 片段， 如果兼容 `pushState` 的浏览器访问了某个 URL 猫点，将会被透明的转换为真实的 URL。 注意使用真实的 URLs 需要 web 服务器支持直接渲染那些页面，因此后端程序也需要做修改。 例如，如果有这样一个路由 `/document/100` ，如果浏览器直接访问它， web 服务器必须能够处理该页面。 趋于对搜索引擎爬虫的兼容，让服务器完全为该页面生成静态 HTML 是非常好的做法 ... 但是如果要做的是一个 web 应用，只需要利用 Javascript 和 Backbone 视图将服务器返回的 REST 数据渲染就很好了。

**start** `Backbone.history.start([options])` 当所有的 Routers 创建并设置完毕，调用 `Backbone.history.start()` 开始监控 `hashchange` 事件并分配路由。后续调用 `Backbone.history.start()` 会抛出一个错误， 并且 `Backbone.History.started` 是一个布尔值，指示是否已经被调用。

需要指出的是，如果想在应用中使用 HTML5 支持的 `pushState` ，只需要这样做： `Backbone.history.start({pushState : true})` 。如果你想使用 `pushState` 的话， 对于那些本身不支持它的浏览器，需要用整页刷新代替， 您可以添加 `{hashChange: false}` 到选项。

如果应用不是基于域名的根路径 `/` ，需要告诉 History 基于什么路径： `Backbone.history.start({pushState: true, root: "/public/search/"})`

当执行后，如果某个路由成功匹配当前 URL， `Backbone.history.start()` 返回 `true` 。 如果没有定义的路由匹配当前 URL，返回 `false` 。

如果服务器已经渲染了整个页面，但又不希望开始 History 时触发初始路由，传入 `silent : true` 即可。

因为在Internet Explorer中基于hash的历史记录依赖于 `<iframe>` ，因此需要确定DOM已准备就绪后再调用 `start()` 。

```
$(function(){
  new WorkspaceRouter();
  new HelpPaneRouter();
  Backbone.history.start({pushState: true});
});
```

# Backbone.sync（同步）

**Backbone.sync** 是 Backbone 每次向服务器读取或保存模型时都要调用执行的函数。 默认情况下，它使用 `jQuery.ajax` 方法发送 RESTful json 请求，并且返回一个 jqXHR。 如果想采用不同的持久化方案，比如 WebSockets, XML, 或 Local Storage，我们可以重载该函数。

**Backbone.sync** 的语法为 `sync(method, model, [options])` 。

- **method** – CRUD 方法 ( `"create"` , `"read"` , `"update"` , or `"delete"` )
- **model** – 要被保存的模型（或要被读取的集合）
- **options** – 成功和失败的回调函数，以及所有 jQuery 请求支持的选项

默认情况下，当 **Backbone.sync** 发送请求以保存模型时，其属性会被序列化为 JSON，并以 `application/json` 的内容类型发送。 当接收到来自服务器的 JSON 响应后，对经过服务器改变的模型进行拆解，然后在客户端更新。 当 `"read"` 请求从服务器端响应一个集合（Collection#fetch）时，便拆解模型属性对象的数组。

当一个模型或集合开始 **sync**到服务器时，将触发一个 `"request"` 事件。 如果请求成功完成，你会得到一个 `"sync"` 事件， 如果请求失败，你会得到一个 `"error"` 事件。

**sync**函数可重写为全局性的 `Backbone.sync` ， 或在细粒度级别， 通过添加一个 `sync` 函数 到Backbone集合或单个模型时。

默认 **sync** 映射 REST 风格的 CRUD 类似下面这样：

- **create → POST** `/collection`
- **read → GET** `/collection[/id]`
- **update → PUT** `/collection/id`
- **patch → PATCH** `/collection/id`
- **delete → DELETE** `/collection/id`

举个例子，一个Rail 4 处理程序响应一个来自 `Backbone` 的 `"update"` 调用，可能是这样的：（在真正的代码中， 千万不要盲目的使用 *`update_attributes`* ， ，你可以被改变的属性始终是白名单。）

```
def update
  account = Account.find params[:id]
  account.update_attributes params.require(:account).permit(:name, :otherparam)
  render :json => account
end
```

一个技巧： 通过设置 `ActiveRecord::Base.include_root_in_json = false` ，在模型上禁用默认命名空间的 `to_json` 来整合 Rails 3.1之前的版本， 。

**ajax** `Backbone.ajax = function(request) { ... };` 如果你想使用自定义的AJAX功能， 或者你的客户端不支持的jQuery.ajax API，你需要调整的东西， 您可以通过设置 `Backbone.ajax` 这样做。

**emulateHTTP** `Backbone.emulateHTTP = true` 如果你想在不支持Backbone的默认REST/HTTP方式的Web服务器上工作， 您可以选择开启 `Backbone.emulateHTTP` 。设置该选项将通过 `POST` 方法伪造 `PUT` ， `PATCH` 和 `DELETE` 请求 用真实的方法设定 `X-HTTP-Method-Override` 头信息。 如果支持 `emulateJSON` ，此时该请求会向服务器传入名为 `_method` 的参数。

```
Backbone.emulateHTTP = true;

model.save();  // POST to "/collection/id", with "_method=PUT" + header.
```

**emulateJSON** `Backbone.emulateJSON = true` 如果你想在不支持发送 `application/json` 编码请求的Web服务器上工作，设置 `Backbone.emulateJSON = true;` 将导致JSON根据模型参数进行序列化， 并通过 `application/x-www-form-urlencoded` MIME类型来发送一个伪造HTML表单请求，

# Backbone.View（视图）

Backbone 视图几乎约定比他们的代码多 — 他们并不限定你的HTML或CSS， 并可以配合使用任何JavaScript模板库。 一般是组织您的接口转换成逻辑视图， 通过模型的支持， 模型变化时， 每一个都可以独立地进行更新， 而不必重新绘制该页面。我们再也不必钻进 JSON 对象中，查找 DOM 元素，手动更新 HTML 了，通过绑定视图的 `render` 函数到模型的 `"change"` 事件 — 模型数据会即时的显示在 UI 中。

**extend** `Backbone.View.extend(properties, [classProperties])` 开始创建自定义的视图类。 通常我们需要重载 render 函数，声明 events， 以及通过 `tagName` , `className` , 或 `id` 为视图指定根元素。

```
var DocumentRow = Backbone.View.extend({

  tagName: "li",

  className: "document-row",

  events: {
    "click .icon":          "open",
    "click .button.edit":   "openEditDialog",
    "click .button.delete": "destroy"
  },

  initialize: function() {
    this.listenTo(this.model, "change", this.render);
  },

  render: function() {
    ...
  }

});
```

直到运行时， 像 `tagName` , `id` , `className` , `el` , 和 `events` 这样的属性也可以被定义为一个函数，

**constructor / initialize** `new View([options])` 有几个特殊的选项， 如果传入，则直接注册到视图中去： `model` , `collection` , `el` , `id` , `className` , `tagName` , `attributes` 和 `events` 。 如果视图定义了一个**initialize**初始化函数， 首先创建视图时，它会立刻被调用。 如果希望创建一个指向 DOM 中已存在的元素的视图，传入该元素作为选项： `new View({el: existingElement})` 。

```
var doc = documents.first();

new DocumentRow({
  model: doc,
  id: "document-row-" + doc.id
});
```

**el** `view.el` 所有的视图都拥有一个 DOM 元素（**el** 属性），即使该元素仍未插入页面中去。视图可以在任何时候渲染，然后一次性插入 DOM 中去，这样能尽量减少 reflows 和 repaints 从而获得高性能的 UI 渲染。 `this.el` 可以从视图的 `tagName` , `className` , `id` 和 `attributes` 创建，如果都未指定，**el** 会是一个空 `div` 。

```
var ItemView = Backbone.View.extend({
  tagName: 'li'
});

var BodyView = Backbone.View.extend({
  el: 'body'
});

var item = new ItemView();
var body = new BodyView();

alert(item.el + ' ' + body.el);
```

**$el** `view.$el` 一个视图元素的缓存jQuery对象。 一个简单的引用，而不是重新包装的DOM元素。

```
view.$el.show();

listView.$el.append(itemView.el);
```

**setElement** `view.setElement(element)` 如果你想应用一个Backbone视图到不同的DOM元素， 使用**setElement**， 这也将创造缓存 `$el` 引用，视图的委托事件从旧元素移动到新元素上。

**attributes** `view.attributes` 属性的键值对， 将被设置为视图 `el` 上的HTML DOM元素的属性， 或者是返回这样的键值对的一个函数。

**$ (jQuery)** `view.$(selector)` 如果页面中引入了 jQuery，每个视图都将拥有 **$** 函数，可以在视图元素查询作用域内运行。 如果使用该作用域内的 jQuery 函数，就不需要从列表中指定的元素获取模型的 ids 这种查询了，我们可以更多的依赖 HTML class 属性。 它等价于运行： `view.$el.find(selector)` 。

```
ui.Chapter = Backbone.View.extend({
  serialize : function() {
    return {
      title: this.$(".title").text(),
      start: this.$(".start-page").text(),
      end:   this.$(".end-page").text()
    };
  }
});
```

**template** `view.template([data])` 虽然模板化的视图 不是Backbone直接提供的一个功能， 它往往是一个在你视图定义**template**函数很好的约定。 如此， 渲染你的视图时， 您方便地访问实例数据。 例如，使用Underscore的模板：

```
var LibraryView = Backbone.View.extend({
  template: _.template(...)
});
```

**render** `view.render()` **render** 默认实现是没有操作的。 重载本函数可以实现从模型数据渲染视图模板，并可用新的 HTML 更新 `this.el` 。 推荐的做法是在 **render** 函数的末尾 `return this` 以开启链式调用。

```
var Bookmark = Backbone.View.extend({
  template: _.template(...),
  render: function() {
    this.$el.html(this.template(this.model.attributes));
    return this;
  }
});
```

Backbone并不知道您首选HTML模板的方法。 **render**（渲染） 函数中可以采用拼接HTML字符串，， 或者使用 `document.createElement` 生成DOM树。 但还是建议选择一个好的 Javascript 模板引擎。 Mustache.js, Haml-js, 和 Eco 都是很好的选择。 因为Underscore.js已经引入页面了，如果你喜欢简单的插入JavaScript的样式模板。 _.template可以使用并是一个很好的选择。

无论基于什么考虑，都永远不要在 Javascript 中拼接 HTML 字符串。 在DocumentCloud中，我们使用Jammit 来打包JavaScript模板，并存储在 `/app/views` 中，作为我们主要的 `core.js` 包的一部分。

**remove** `view.remove()` 从 DOM 中移除一个视图。同事调用stopListening来移除通过listenTo绑定在视图上的 所有事件。

**delegateEvents** `delegateEvents([events])` 采用 jQuery 的 `on` 函数来为视图内的 DOM 事件提供回调函数声明。 如果未传入 **events** 对象，使用 `this.events` 作为事件源。 事件对象的书写格式为 `{"event selector": "callback"}` 。 省略 `selector` 则事件被绑定到视图的根元素（ `this.el` ）。 默认情况下， `delegateEvents` 会在视图的构造函数内被调用，因此如果有 `events` 对象，所有的 DOM 事件已经被连接， 并且我们永远不需要去手动调用本函数。

`events` 属性也可以被定义成返回 **events** 对象的函数，这样让我们定义事件，以及实现事件的继承变得更加方便。

视图 render 期间使用 **delegateEvents** 相比用 jQuery 向子元素绑定事件有更多优点。 所有注册的函数在传递给 jQuery 之前已被绑定到视图上，因此当回调函数执行时， `this` 仍将指向视图对象。 当 **delegateEvents** 再次运行，此时或许需要一个不同的 `events` 对象，所以所有回调函数将被移除，然后重新委托 — 这对模型不同行为也不同的视图挺有用处。

搜索结果页面显示文档的视图看起来类似这样：

```
var DocumentView = Backbone.View.extend({

  events: {
    "dblclick"                : "open",
    "click .icon.doc"         : "select",
    "contextmenu .icon.doc"   : "showMenu",
    "click .show_notes"       : "toggleNotes",
    "click .title .lock"      : "editAccessLevel",
    "mouseover .title .date"  : "showTooltip"
  },

  render: function() {
    this.$el.html(this.template(this.model.attributes));
    return this;
  },

  open: function() {
    window.open(this.model.get("viewer_url"));
  },

  select: function() {
    this.model.set({selected: true});
  },

  ...

});
```

**undelegateEvents** `undelegateEvents()` 删除视图所有委托事件。如果要从临时的DOM中禁用或删除视图时，比较有用。

# Utility（实用功能）

**Backbone.noConflict** `var backbone = Backbone.noConflict();` 返回 `Backbone` 对象的原始值。您可以使用`Backbone.noConflict()`的返回值以保持局部引用Backbone。通常用于在第三方网站上引入了多个 Backbone 文件，避免冲突。

```
var localBackbone = Backbone.noConflict();
var model = localBackbone.Model.extend(...);
```

**Backbone.$** `Backbone.$ = $;` 如果页面上有多个`jQuery`副本，或者只是想告诉Backbone使用特定对象作为其DOM / Ajax库，那么这个属性可以为您服务。如果您正在使用CommonJS加载Backbone （例如，节点，组件，或browserify） 您必须手动设置该属性。

```
var Backbone.$ = require('jquery');
```

# F.A.Q.

**Why use Backbone, not [other framework X]?** If your eye hasn't already been caught by the adaptability and elan on display in the above list of examples, we can get more specific: Backbone.js aims to provide the common foundation that data-rich web applications with ambitious interfaces require — while very deliberately avoiding painting you into a corner by making any decisions that you're better equipped to make yourself.

- The focus is on supplying you with helpful methods to manipulate and query your data, not on HTML widgets or reinventing the JavaScript object model.
- Backbone does not force you to use a single template engine. Views can bind to HTML constructed in your favorite way.
- It's smaller. There are fewer kilobytes for your browser or phone to download, and less *conceptual* surface area. You can read and understand the source in an afternoon.
- It doesn't depend on stuffing application logic into your HTML. There's no embedded JavaScript, template logic, or binding hookup code in `data-` or `ng-` attributes, and no need to invent your own HTML tags.
- Synchronous events are used as the fundamental building block, not a difficult-to-reason-about run loop, or by constantly polling and traversing your data structures to hunt for changes. And if you want a specific event to be asynchronous and aggregated, no problem.
- Backbone scales well, from embedded widgets to massive apps.
- Backbone is a library, not a framework, and plays well with others. You can embed Backbone widgets in Dojo apps without trouble, or use Backbone models as the data backing for D3 visualizations (to pick two entirely random examples).
- "Two-way data-binding" is avoided. While it certainly makes for a nifty demo, and works for the most basic CRUD, it doesn't tend to be terribly useful in your real-world app. Sometimes you want to update on every keypress, sometimes on blur, sometimes when the panel is closed, and sometimes when the "save" button is clicked. In almost all cases, simply serializing the form to JSON is faster and easier. All that aside, if your heart is set, go for it.
- There's no built-in performance penalty for choosing to structure your code with Backbone. And if you do want to optimize further, thin models and templates with flexible granularity make it easy to squeeze every last drop of potential performance out of, say, IE8.

**There's More Than One Way To Do It** It's common for folks just getting started to treat the examples listed on this page as some sort of gospel truth. In fact, Backbone.js is intended to be fairly agnostic about many common patterns in client-side code. For example...

**References between Models and Views** can be handled several ways. Some people like to have direct pointers, where views correspond 1:1 with models ( `model.view` and `view.model` ). Others prefer to have intermediate "controller" objects that orchestrate the creation and organization of views into a hierarchy. Others still prefer the evented approach, and always fire events instead of calling methods directly. All of these styles work well.

**Batch operations** on Models are common, but often best handled differently depending on your server-side setup. Some folks don't mind making individual Ajax requests. Others create explicit resources for RESTful batch operations: `/notes/batch/destroy?ids=1,2,3,4` . Others tunnel REST over JSON, with the creation of "changeset" requests:

```
{
  "create":  [array of models to create]
  "update":  [array of models to update]
  "destroy": [array of model ids to destroy]
}
```

**Feel free to define your own events.** Backbone.Events is designed so that you can mix it in to any JavaScript object or prototype. Since you can use any string as an event, it's often handy to bind and trigger your own custom events: `model.on("selected:true")` or `model.on("editing")`

**Render the UI** as you see fit. Backbone is agnostic as to whether you use Underscore templates, Mustache.js, direct DOM manipulation, server-side rendered snippets of HTML, or jQuery UI in your `render` function. Sometimes you'll create a view for each model ... sometimes you'll have a view that renders thousands of models at once, in a tight loop. Both can be appropriate in the same app, depending on the quantity of data involved, and the complexity of the UI.

**Nested Models & Collections** It's common to nest collections inside of models with Backbone. For example, consider a `Mailbox` model that contains many `Message` models. One nice pattern for handling this is have a `this.messages` collection for each mailbox, enabling the lazy-loading of messages, when the mailbox is first opened ... perhaps with `MessageList` views listening for `"add"` and `"remove"` events.

```
var Mailbox = Backbone.Model.extend({

  initialize: function() {
    this.messages = new Messages;
    this.messages.url = '/mailbox/' + this.id + '/messages';
    this.messages.on("reset", this.updateCounts);
  },

  ...

});

var inbox = new Mailbox;

// And then, when the Inbox is opened:

inbox.messages.fetch({reset: true});
```

If you're looking for something more opinionated, there are a number of Backbone plugins that add sophisticated associations among models, available on the wiki.

Backbone doesn't include direct support for nested models and collections or "has many" associations because there are a number of good patterns for modeling structured data on the client side, and *Backbone should provide the foundation for implementing any of them.* You may want to…

- Mirror an SQL database's structure, or the structure of a NoSQL database.
- Use models with arrays of "foreign key" ids, and join to top level collections (a-la tables).
- For associations that are numerous, use a range of ids instead of an explicit list.
- Avoid ids, and use direct references, creating a partial object graph representing your data set.
- Lazily load joined models from the server, or lazily deserialize nested models from JSON documents.

**Loading Bootstrapped Models** When your app first loads, it's common to have a set of initial models that you know you're going to need, in order to render the page. Instead of firing an extra AJAX request to fetch them, a nicer pattern is to have their data already bootstrapped into the page. You can then use reset to populate your collections with the initial data. At DocumentCloud, in the ERB template for the workspace, we do something along these lines:

```
<script>
  var accounts = new Backbone.Collection;
  accounts.reset(<%= @accounts.to_json %>);
  var projects = new Backbone.Collection;
  projects.reset(<%= @projects.to_json(:collaborators => true) %>);
</script>
```

You have to escape `</` within the JSON string, to prevent javascript injection attacks.

**Extending Backbone** Many JavaScript libraries are meant to be insular and self-enclosed, where you interact with them by calling their public API, but never peek inside at the guts. Backbone.js is *not* that kind of library.

Because it serves as a foundation for your application, you're meant to extend and enhance it in the ways you see fit — the entire source code is annotated to make this easier for you. You'll find that there's very little there apart from core functions, and most of those can be overridden or augmented should you find the need. If you catch yourself adding methods to `Backbone.Model.prototype`, or creating your own base subclass, don't worry — that's how things are supposed to work.

**How does Backbone relate to "traditional" MVC?** Different implementations of the Model-View-Controller pattern tend to disagree about the definition of a controller. If it helps any, in Backbone, the View class can also be thought of as a kind of controller, dispatching events that originate from the UI, with the HTML template serving as the true view. We call it a View because it represents a logical chunk of UI, responsible for the contents of a single DOM element.

Comparing the overall structure of Backbone to a server-side MVC framework like **Rails**, the pieces line up like so:

- **Backbone.Model** – Like a Rails model minus the class methods. Wraps a row of data in business logic.
- **Backbone.Collection** – A group of models on the client-side, with sorting/filtering/aggregation logic.
- **Backbone.Router** – Rails `routes.rb` + Rails controller actions. Maps URLs to functions.
- **Backbone.View** – A logical, re-usable piece of UI. Often, but not always, associated with a model.
- **Client-side Templates** – Rails `.html.erb` views, rendering a chunk of HTML.

**Binding "this"** Perhaps the single most common JavaScript "gotcha" is the fact that when you pass a function as a callback, its value for `this` is lost. When dealing with events and callbacks in Backbone, you'll often find it useful to rely on listenTo or the optional `context` argument that many of Underscore and Backbone's methods use to specify the `this` that will be used when the callback is later invoked. (See _.each, _.map, and object.on, to name a few). View events are automatically bound to the view's context for you. You may also find it helpful to use _.bind and _.bindAll from Underscore.js.

```
var MessageList = Backbone.View.extend({

  initialize: function() {
    var messages = this.collection;
    messages.on("reset", this.render, this);
    messages.on("add", this.addMessage, this);
    messages.on("remove", this.removeMessage, this);

    messsages.each(this.addMessage, this);
  }

});

// Later, in the app...

Inbox.messages.add(newMessage);
```

**Working with Rails** Backbone.js was originally extracted from a Rails application; getting your client-side (Backbone) Models to sync correctly with your server-side (Rails) Models is painless, but there are still a few things to be aware of.

By default, Rails versions prior to 3.1 add an extra layer of wrapping around the JSON representation of models. You can disable this wrapping by setting:

```
ActiveRecord::Base.include_root_in_json = false
```

... in your configuration. Otherwise, override parse to pull model attributes out of the wrapper. Similarly, Backbone PUTs and POSTs direct JSON representations of models, where by default Rails expects namespaced attributes. You can have your controllers filter attributes directly from `params`, or you can override toJSON in Backbone to add the extra wrapping Rails expects.

# Examples

The list of examples that follows, while long, is not exhaustive. If you've worked on an app that uses Backbone, please add it to the wiki page of Backbone apps.

Jérôme Gravel-Niquet has contributed a Todo List application that is bundled in the repository as Backbone example. If you're wondering where to get started with Backbone in general, take a moment to read through the annotated source. The app uses a LocalStorage adapter to transparently save all of your todos within your browser, instead of sending them to a server. Jérôme also has a version hosted at localtodos.com.

## DocumentCloud

The DocumentCloud workspace is built on Backbone.js, with *Documents*, *Projects*, *Notes*, and *Accounts* all as Backbone models and collections. If you're interested in history — both Underscore.js and Backbone.js were originally extracted from the DocumentCloud codebase, and packaged into standalone JS libraries.

## USA Today

USA Today takes advantage of the modularity of Backbone's data/model lifecycle — which makes it simple to create, inherit, isolate, and link application objects — to keep the codebase both manageable and efficient. The new website also makes heavy use of the Backbone Router to control the page for both pushState-capable and legacy browsers. Finally, the team took advantage of Backbone's Event module to create a PubSub API that allows third parties and analytics packages to hook into the heart of the app.

## Rdio

New Rdio was developed from the ground up with a component based framework based on Backbone.js. Every component on the screen is dynamically loaded and rendered, with data provided by the Rdio API. When changes are pushed, every component can update itself without reloading the page or interrupting the user's music. All of this relies on Backbone's views and models, and all URL routing is handled by Backbone's Router. When data changes are signaled in realtime, Backbone's Events notify the interested components in the data changes. Backbone forms the core of the new, dynamic, realtime Rdio web and *desktop* applications.

## Hulu

Hulu used Backbone.js to build its next generation online video experience. With Backbone as a foundation, the web interface was rewritten from scratch so that all page content can be loaded dynamically with smooth transitions as you navigate. Backbone makes it easy to move through the app quickly without the reloading of scripts and embedded videos, while also offering models and collections for additional data manipulation support.

## Quartz

Quartz sees itself as a digitally native news outlet for the new global economy. Because Quartz believes in the future of open, cross-platform web applications, they selected Backbone and Underscore to fetch, sort, store, and display content from a custom WordPress API. Although qz.com uses responsive design for phone, tablet, and desktop browsers, it also takes advantage of Backbone events and views to render device-specific templates in some cases.

## Earth

Earth.nullschool.net displays real-time weather conditions on an interactive animated globe, and Backbone provides the foundation upon which all of the site's components are built. Despite the presence of several other javascript libraries, Backbone's non-opinionated design made it effortless to mix-in the Events functionality used for distributing state changes throughout the page. When the decision was made to switch to Backbone, large blocks of custom logic simply disappeared.

## Vox

Vox Media, the publisher of SB Nation, The Verge, Polygon, Eater, Racked, Curbed, and Vox.com, uses Backbone throughout Chorus, its home-grown publishing platform. Backbone powers the liveblogging platform and commenting system used across all Vox Media properties; Coverage, an internal editorial coordination tool; SB Nation Live, a live event coverage and chat tool; and Vox Cards, Vox.com's highlighter-and-index-card inspired app for providing context about the news.

## Gawker Media

Kinja is Gawker Media's publishing platform designed to create great stories by breaking down the lines between the traditional roles of content creators and consumers. Everyone — editors, readers, marketers — have access to the same tools to engage in passionate discussion and pursue the truth of the story. Sharing, recommending, and following within the Kinja ecosystem allows for improved information discovery across all the sites.

Kinja is the platform behind Gawker, Gizmodo, Lifehacker, io9 and other Gawker Media blogs. Backbone.js underlies the front-end application code that powers everything from user authentication to post authoring, commenting, and even serving ads. The JavaScript stack includes Underscore.js and jQuery, with some plugins, all loaded with RequireJS. Closure templates are shared between the Play! Framework based Scala application and Backbone views, and the responsive layout is done with the Foundation framework using SASS.

## Flow

MetaLab used Backbone.js to create Flow, a task management app for teams. The workspace relies on Backbone.js to construct task views, activities, accounts, folders, projects, and tags. You can see the internals under `window.Flow`.

## Gilt Groupe

Gilt Groupe uses Backbone.js to build multiple applications across their family of sites. Gilt's mobile website uses Backbone and Zepto.js to create a blazing-fast shopping experience for users on-the-go, while Gilt Live combines Backbone with WebSockets to display the items that customers are buying in real-time. Gilt's search functionality also uses Backbone to filter and sort products efficiently by moving those actions to the client-side.

## Enigma

Enigma is a portal amassing the largest collection of public data produced by governments, universities, companies, and organizations. Enigma uses Backbone Models and Collections to represent complex data structures; and Backbone's Router gives Enigma users unique URLs for application states, allowing them to navigate quickly through the site while maintaining the ability to bookmark pages and navigate forward and backward through their session.

## NewsBlur

NewsBlur is an RSS feed reader and social news network with a fast and responsive UI that feels like a native desktop app. Backbone.js was selected for a major rewrite and transition from spaghetti code because of its powerful yet simple feature set, easy integration, and large community. If you want to poke around under the hood, NewsBlur is also entirely open-source.

## WordPress.com

WordPress.com is the software-as-a-service version of WordPress. It uses Backbone.js Models, Collections, and Views in its Notifications system. Backbone.js was selected because it was easy to fit into the structure of the application, not the other way around. Automattic (the company behind WordPress.com) is integrating Backbone.js into the Stats tab and other features throughout the homepage.

## Foursquare

Foursquare is a fun little startup that helps you meet up with friends, discover new places, and save money. Backbone Models are heavily used in the core JavaScript API layer and Views power many popular features like the homepage map and lists.

## Bitbucket

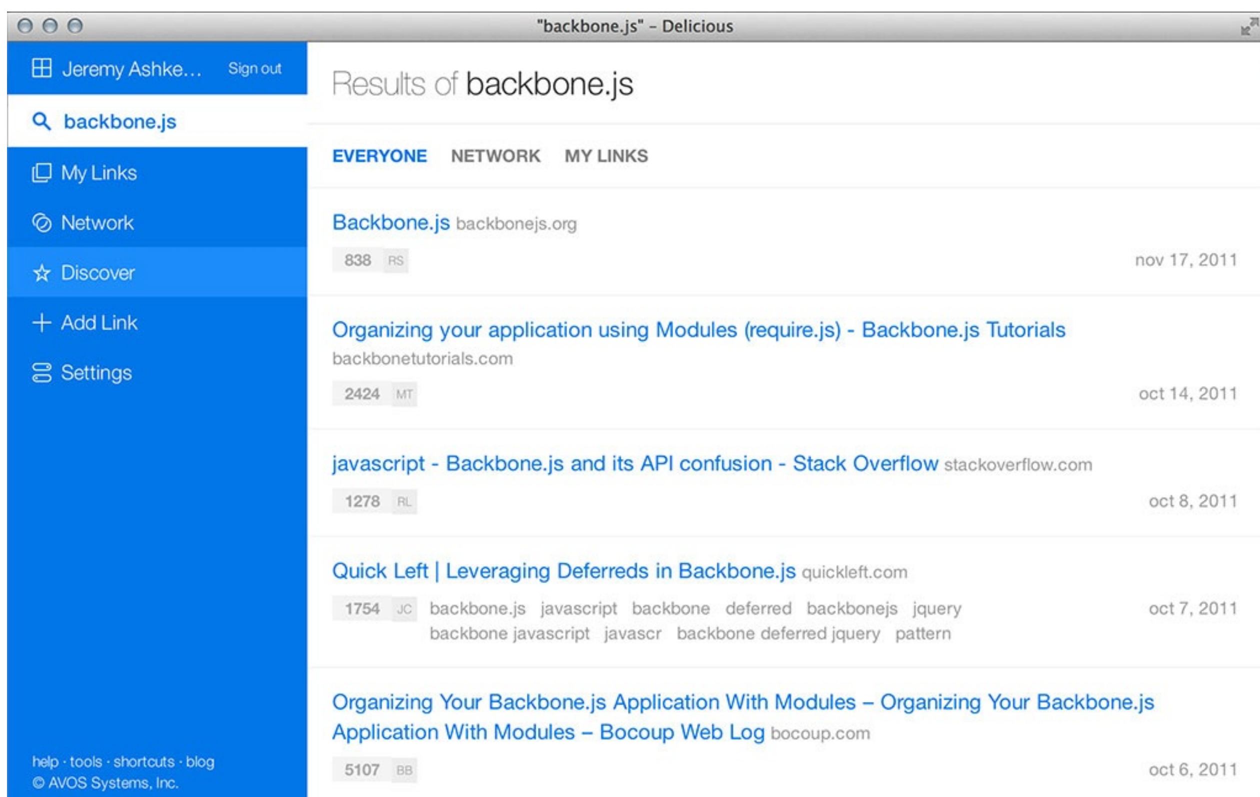
Bitbucket is a free source code hosting service for Git and Mercurial. Through its models and collections, Backbone.js has proved valuable in supporting Bitbucket's REST API, as well as newer components such as in-line code comments and approvals for pull requests. Mustache templates provide server and client-side rendering, while a custom Google Closure inspired life-cycle for widgets allows Bitbucket to decorate existing DOM trees and insert new ones.

## Disqus

Disqus chose Backbone.js to power the latest version of their commenting widget. Backbone’s small footprint and easy extensibility made it the right choice for Disqus’ distributed web application, which is hosted entirely inside an iframe and served on thousands of large web properties, including IGN, Wired, CNN, MLB, and more.

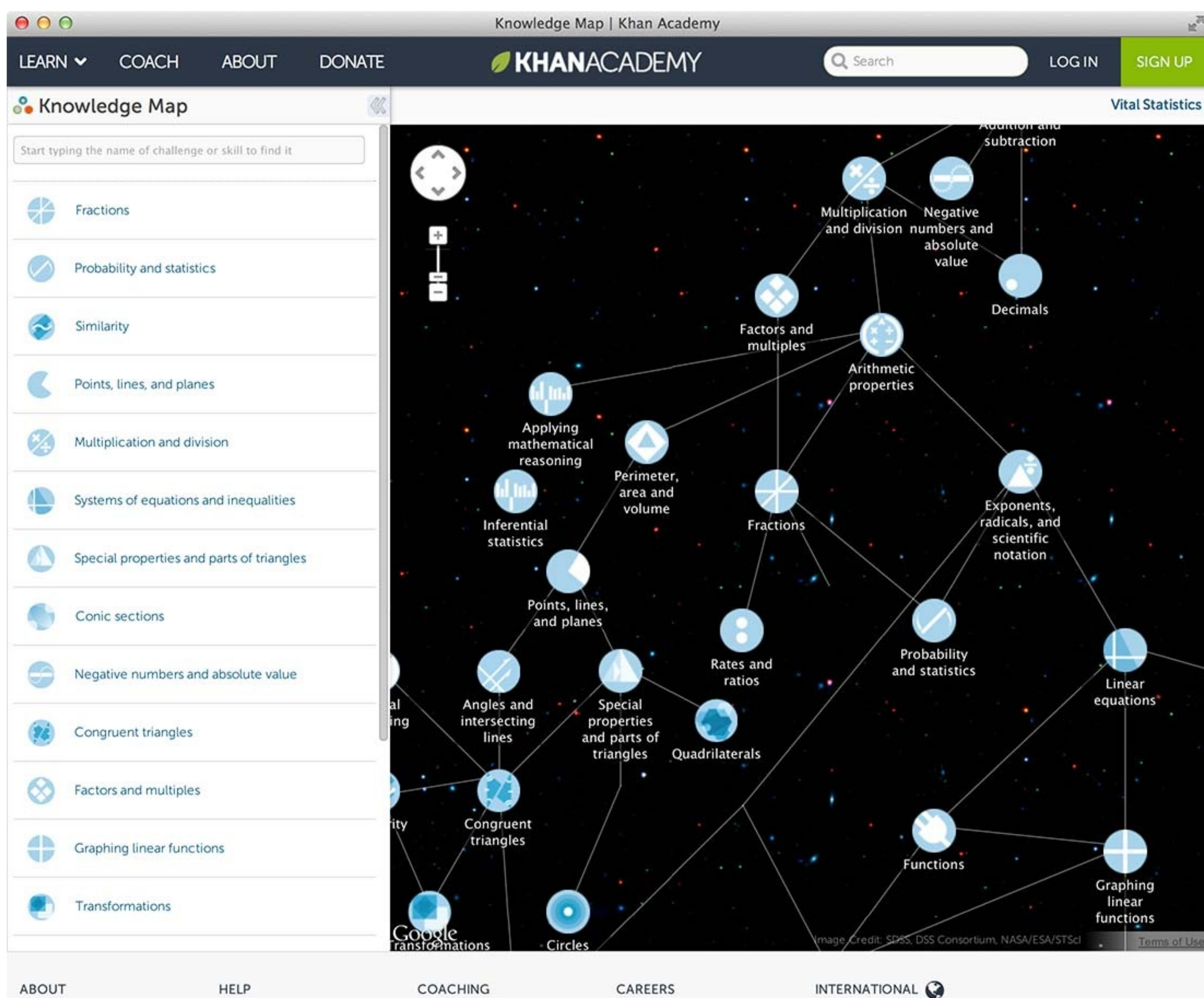
## Delicious

Delicious is a social bookmarking platform making it easy to save, sort, and store bookmarks from across the web. Delicious uses Chaplin.js, Backbone.js and AppCache to build a full-featured MVC web app. The use of Backbone helped the website and mobile apps share a single API service, and the reuse of the model tier made it significantly easier to share code during the recent Delicious redesign.



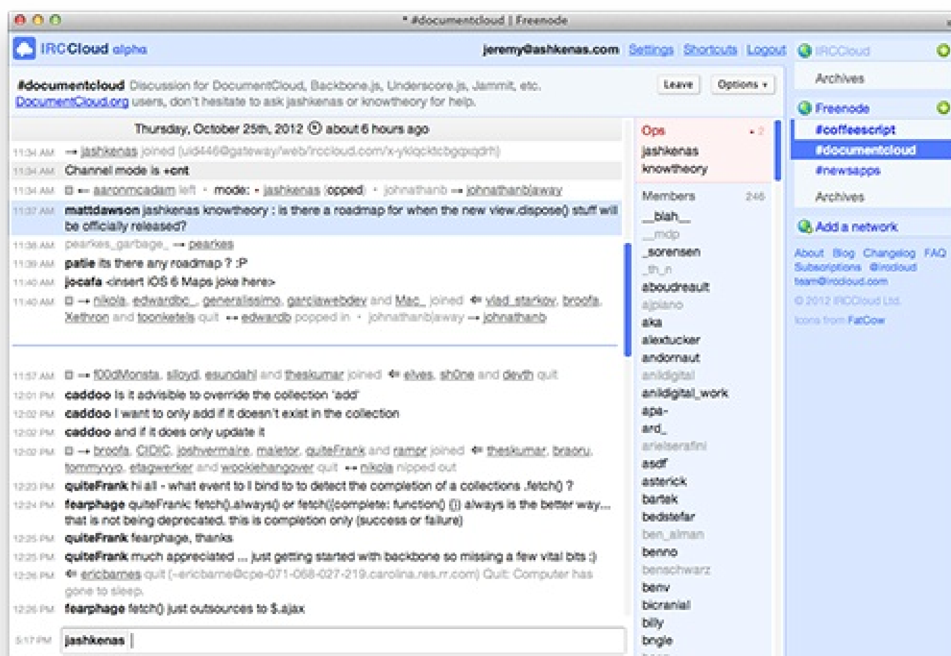

## Khan Academy

Khan Academy is on a mission to provide a free world-class education to anyone anywhere. With thousands of videos, hundreds of JavaScript-driven exercises, and big plans for the future, Khan Academy uses Backbone to keep frontend code modular and organized. User profiles and goal setting are implemented with Backbone, jQuery and Handlebars, and most new feature work is being pushed to the client side, greatly increasing the quality of the API.



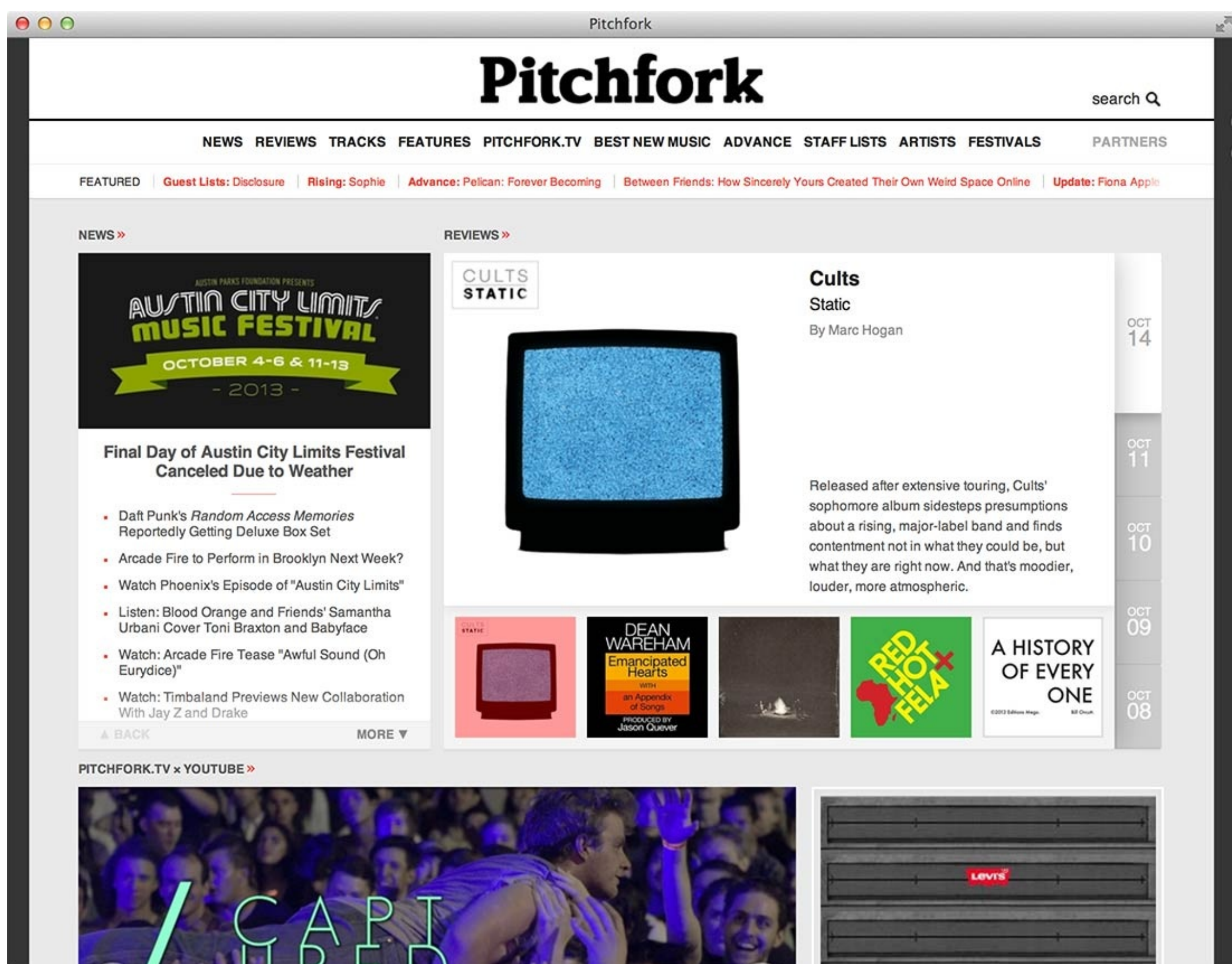

## IRCCloud

IRCCloud is an always-connected IRC client that you use in your browser — often leaving it open all day in a tab. The sleek web interface communicates with an Erlang backend via websockets and the IRCCloud API. It makes heavy use of Backbone.js events, models, views and routing to keep your IRC conversations flowing in real time.



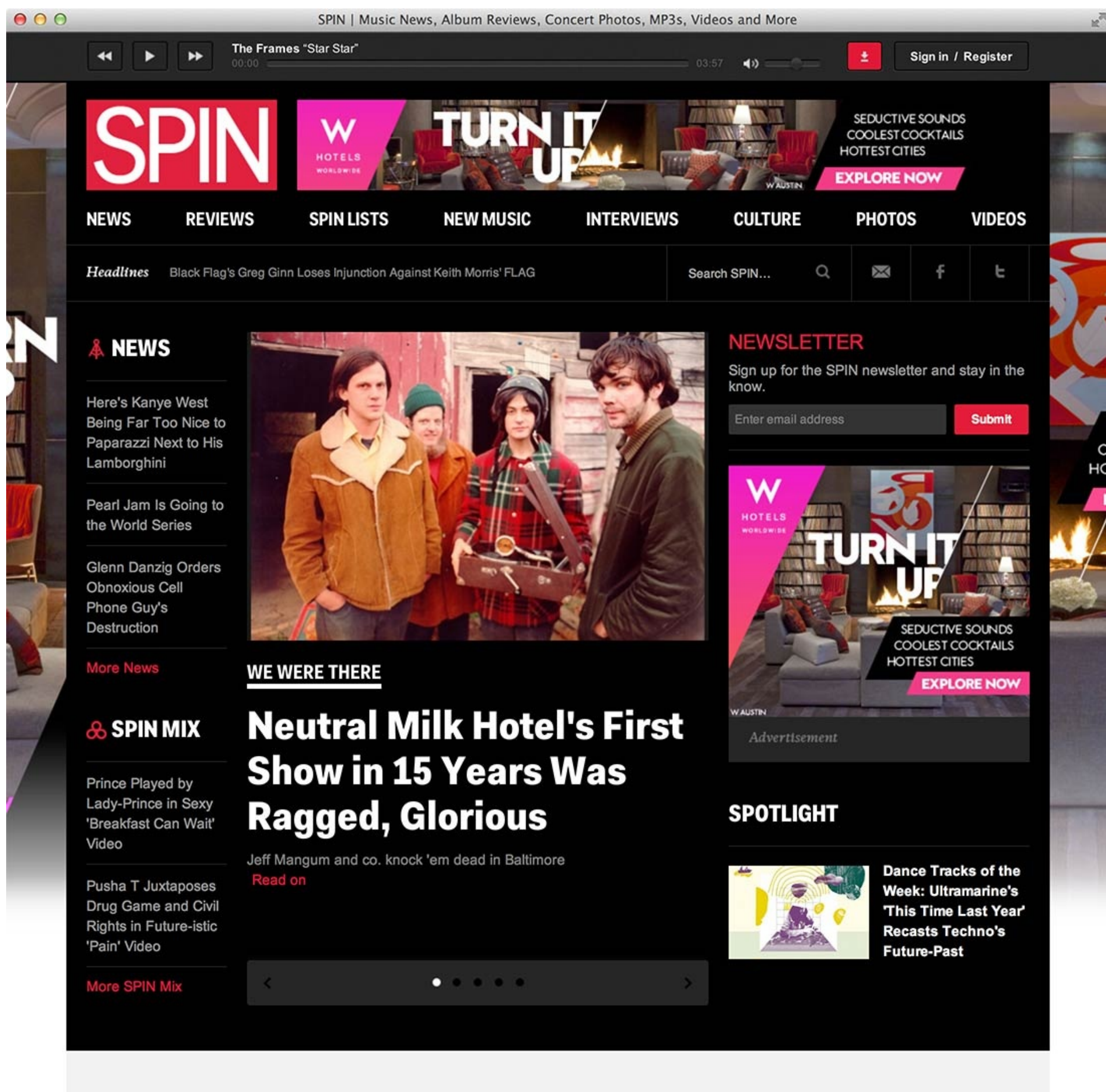

## Pitchfork

Pitchfork uses Backbone.js to power its site-wide audio player, Pitchfork.tv, location routing, a write-thru page fragment cache, and more. Backbone.js (and Underscore.js) helps the team create clean and modular components, move very quickly, and focus on the site, not the spaghetti.




## Spin

Spin pulls in the latest news stories from their internal API onto their site using Backbone models and collections, and a custom `sync` method. Because the music should never stop playing, even as you click through to different "pages", Spin uses a Backbone router for navigation within the site.




## ZocDoc

ZocDoc helps patients find local, in-network doctors and dentists, see their real-time availability, and instantly book appointments. On the public side, the webapp uses Backbone.js to handle client-side state and rendering in search pages and doctor profiles. In addition, the new version of the doctor-facing part of the website is a large single-page application that benefits from Backbone's structure and modularity. ZocDoc's Backbone classes are tested with Jasmine, and delivered to the end user with Cassette.



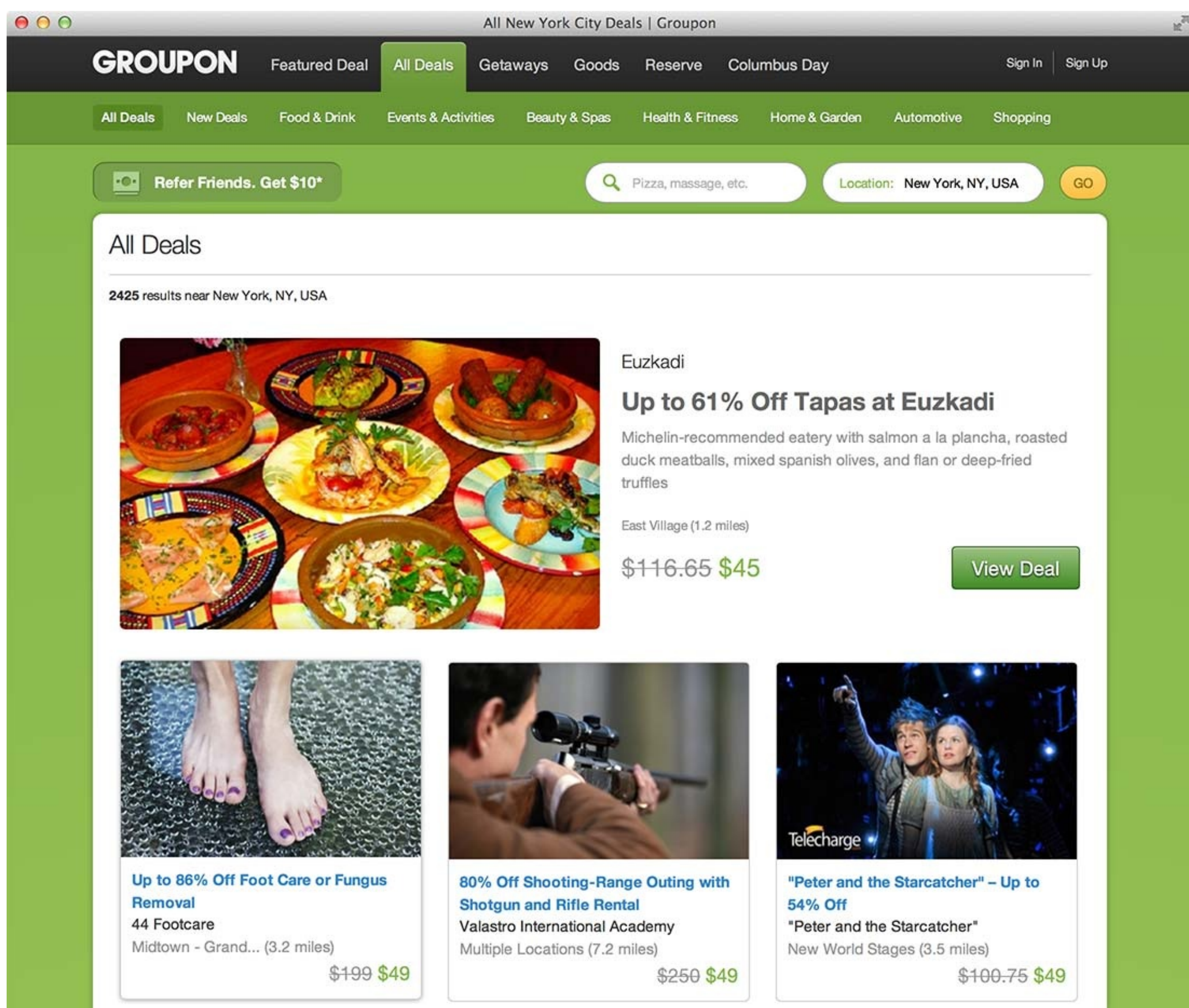

## Walmart Mobile

Walmart used Backbone.js to create the new version of their mobile web application and created two new frameworks in the process. Thorax provides mixins, inheritable events, as well as model and collection view bindings that integrate directly with Handlebars templates. Lumbar allows the application to be split into modules which can be loaded on demand, and creates platform specific builds for the portions of the web application that are embedded in Walmart's native Android and iOS applications.


Carrier
1:32 AM
Walmart
Search Results | Item Details
PS3 500GB Grand Theft Auto V Console Bundle
$269.00
(6)

## Groupon Now!

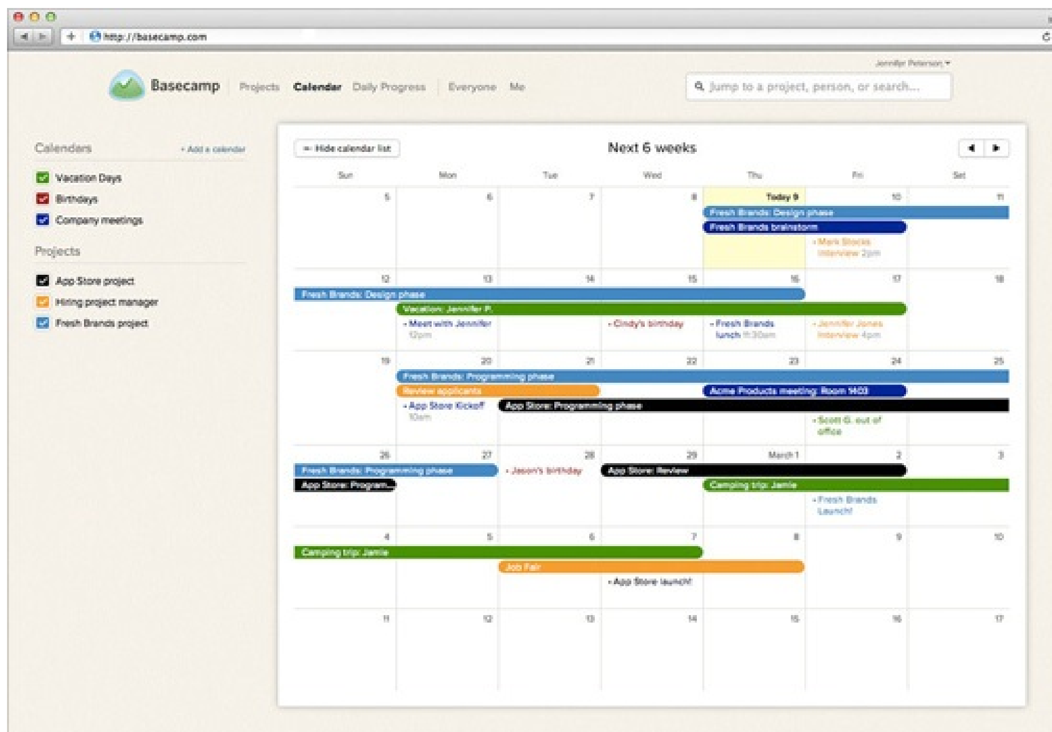
Groupon Now! helps you find local deals that you can buy and use right now. When first developing the product, the team decided it would be AJAX heavy with smooth transitions between sections instead of full refreshes, but still needed to be fully linkable and shareable. Despite never having used Backbone before, the learning curve was incredibly quick — a prototype was hacked out in an afternoon, and the team was able to ship the product in two weeks. Because the source is minimal and understandable, it was easy to add several Backbone extensions for Groupon Now!: changing the router to handle URLs with querystring parameters, and adding a simple in-memory store for caching repeated requests for the same data.




## Basecamp

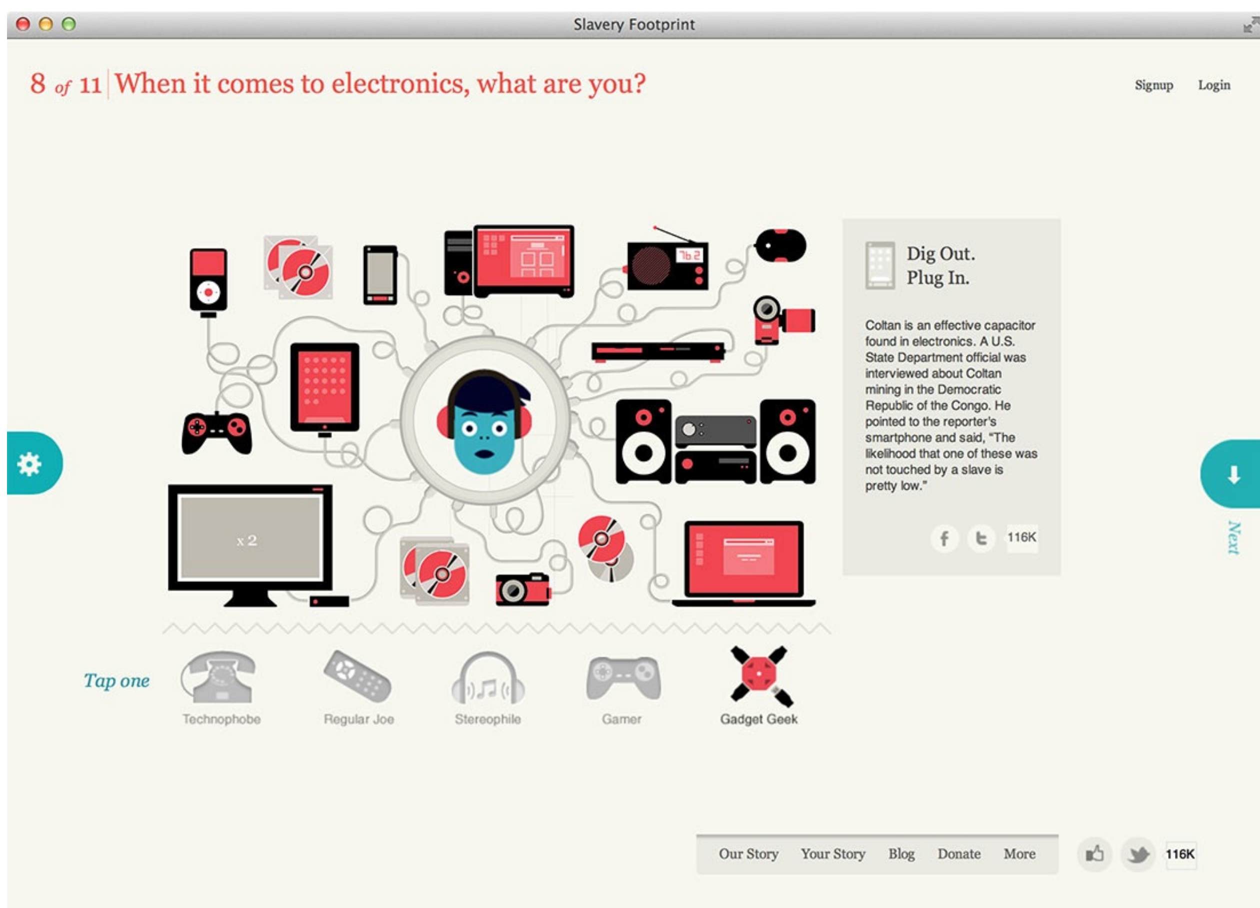
37Signals chose Backbone.js to create the calendar feature of its popular project management software Basecamp. The Basecamp Calendar uses Backbone.js models and views in conjunction with the Eco templating system to present a polished, highly interactive

group scheduling interface.



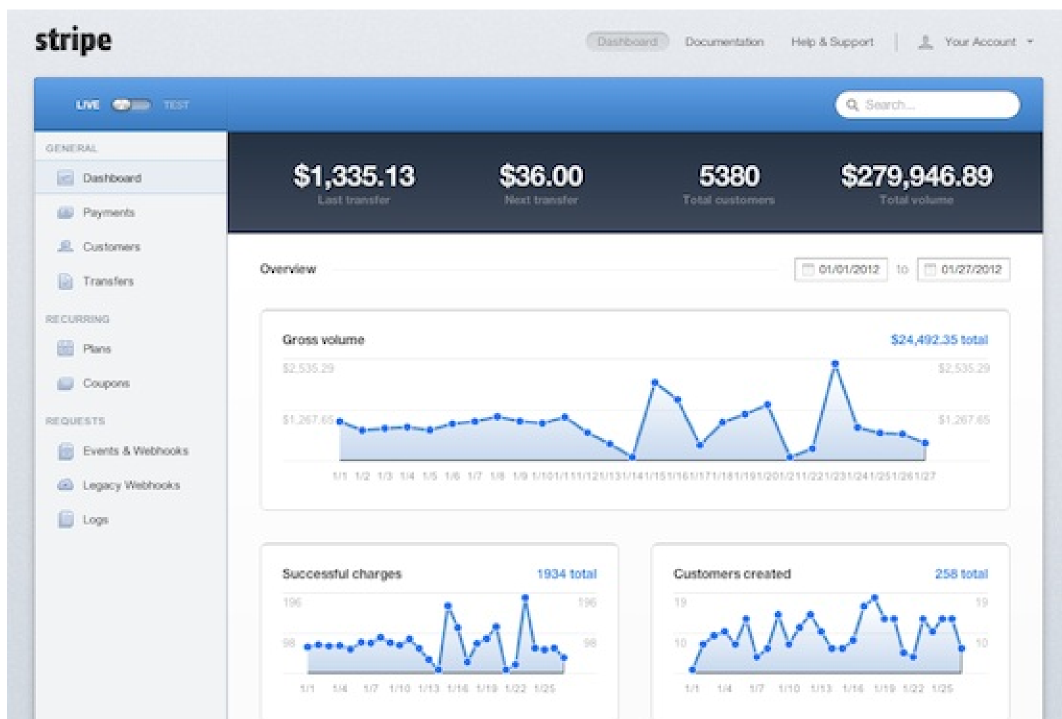

## Slavery Footprint

Slavery Footprint allows consumers to visualize how their consumption habits are connected to modern-day slavery and provides them with an opportunity to have a deeper conversation with the companies that manufacture the goods they purchased. Based in Oakland, California, the Slavery Footprint team works to engage individuals, groups, and businesses to build awareness for and create deployable action against forced labor, human trafficking, and modern-day slavery through online tools, as well as off-line community education and mobilization programs.



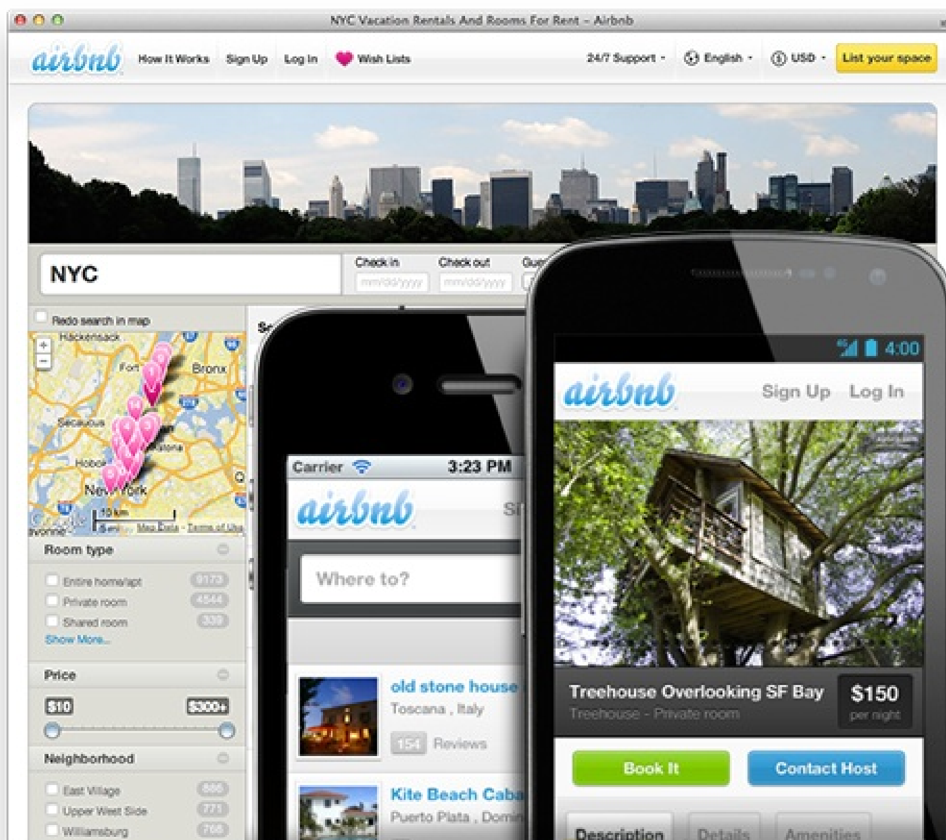

## Stripe

Stripe provides an API for accepting credit cards on the web. Stripe's management interface was recently rewritten from scratch in CoffeeScript using Backbone.js as the primary framework, Eco for templates, Sass for stylesheets, and Stitch to package everything together as CommonJS modules. The new app uses Stripe's API directly for the majority of its actions; Backbone.js models made it simple to map client-side models to their corresponding RESTful resources.



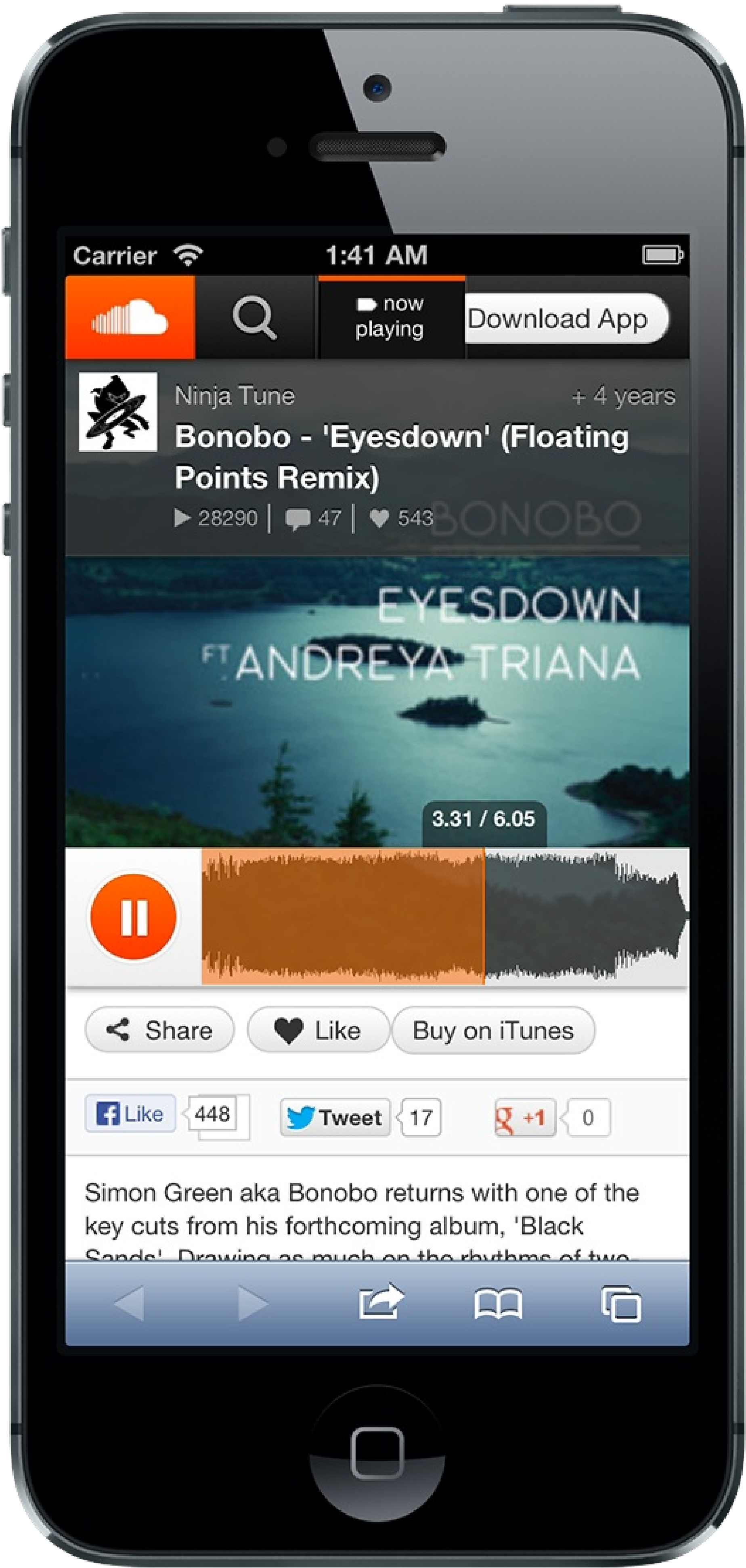

## Airbnb

Airbnb uses Backbone in many of its products. It started with Airbnb Mobile Web (built in six weeks by a team of three) and has since grown to Wish Lists, Match, Search, Communities, Payments, and Internal Tools.




## SoundCloud Mobile

SoundCloud is the leading sound sharing platform on the internet, and Backbone.js provides the foundation for SoundCloud Mobile. The project uses the public SoundCloud API as a data source (channeled through a nginx proxy), jQuery templates for the rendering, Qunit and PhantomJS for the testing suite. The JS code, templates and CSS are built for the production deployment with various Node.js tools like ready.js, Jake, jsdom. The **Backbone.History** was modified to support the HTML5 `history.pushState`. **Backbone.sync** was extended with an additional SessionStorage based cache layer.


Carrier
1:41 AM
now
playing
Download App
Ninja Tune
+ 4 years
Bonobo - 'Eyesdown' (Floating
Points Remix)
28290
47
543
BONOBO
EYESDOWN
FT ANDREYA TRIANA
3.31 / 6.05
Share
Like
Buy on iTunes
Like
448
Tweet
17
+1
0
Simon Green aka Bonobo returns with one of the
key cuts from his forthcoming album, 'Black

## Art.sy

Art.sy is a place to discover art you'll love. Art.sy is built on Rails, using Grape to serve a robust JSON API. The main site is a single page app written in CoffeeScript and uses Backbone to provide structure around this API. An admin panel and partner CMS have also been extracted into their own API-consuming Backbone projects.

## Pandora

When Pandora redesigned their site in HTML5, they chose Backbone.js to help manage the user interface and interactions. For example, there's a model that represents the "currently playing track", and multiple views that automatically update when the current track changes. The station list is a collection, so that when stations are added or changed, the UI stays up to date.

## Inkling

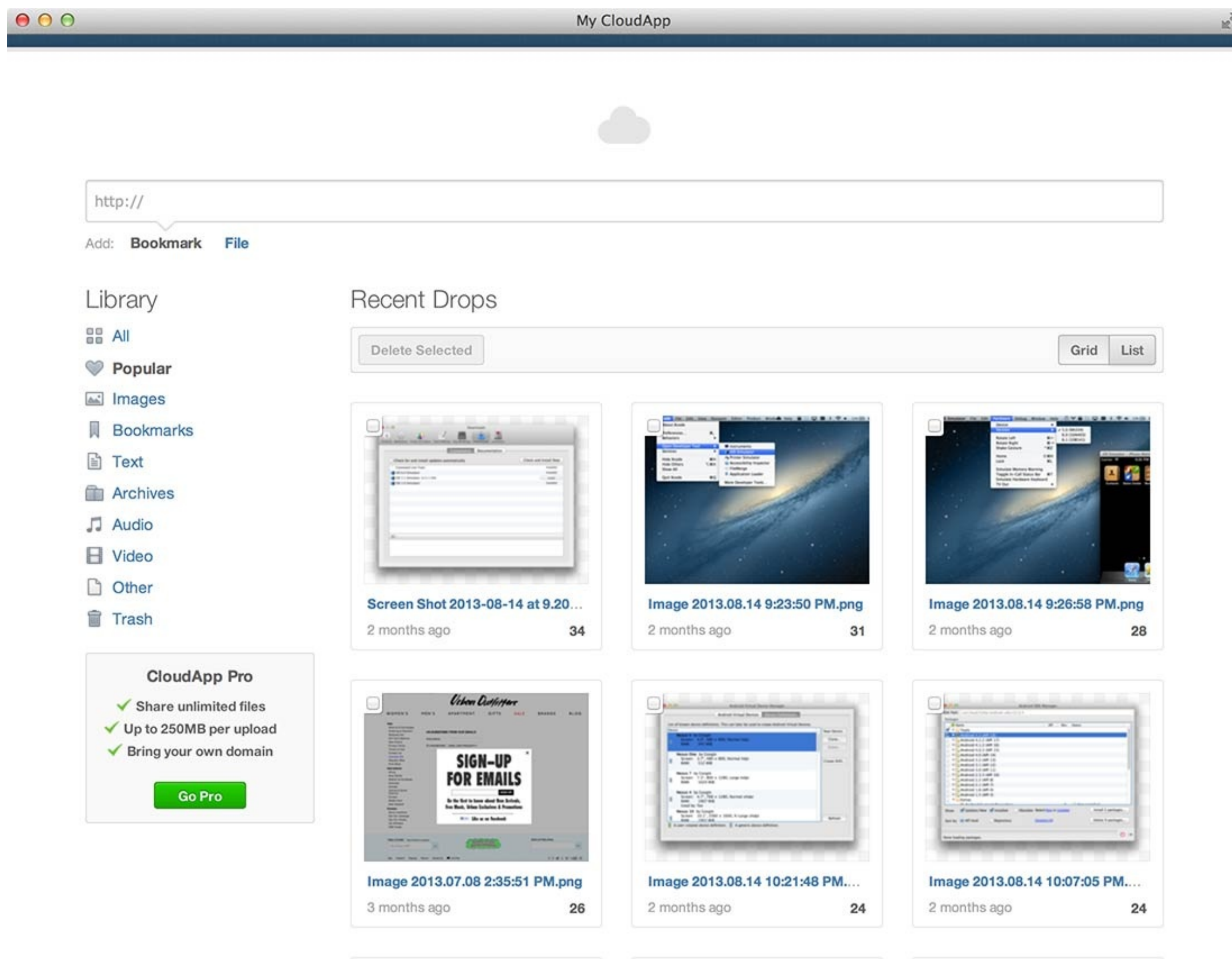
Inkling is a cross-platform way to publish interactive learning content. Inkling for Web uses Backbone.js to make hundreds of complex books — from student textbooks to travel guides and programming manuals — engaging and accessible on the web. Inkling supports WebGL-enabled 3D graphics, interactive assessments, social sharing, and a system for running practice code right in the book, all within a single page Backbone-driven app. Early on, the team decided to keep the site lightweight by using only Backbone.js and raw JavaScript. The result? Complete source code weighing in at a mere 350kb with feature-parity across the iPad, iPhone and web clients. Give it a try with this excerpt from JavaScript: The Definitive Guide.

## Code School

Code School courses teach people about various programming topics like CoffeeScript, CSS, Ruby on Rails, and more. The new Code School course challenge page is built from the ground up on Backbone.js, using everything it has to offer: the router, collections, models, and complex event handling. Before, the page was a mess of jQuery DOM manipulation and manual Ajax calls. Backbone.js helped introduce a new way to think about developing an organized front-end application in JavaScript.

## CloudApp

CloudApp is simple file and link sharing for the Mac. Backbone.js powers the web tools which consume the documented API to manage Drops. Data is either pulled manually or pushed by Pusher and fed to Mustache templates for rendering. Check out the annotated source code to see the magic.

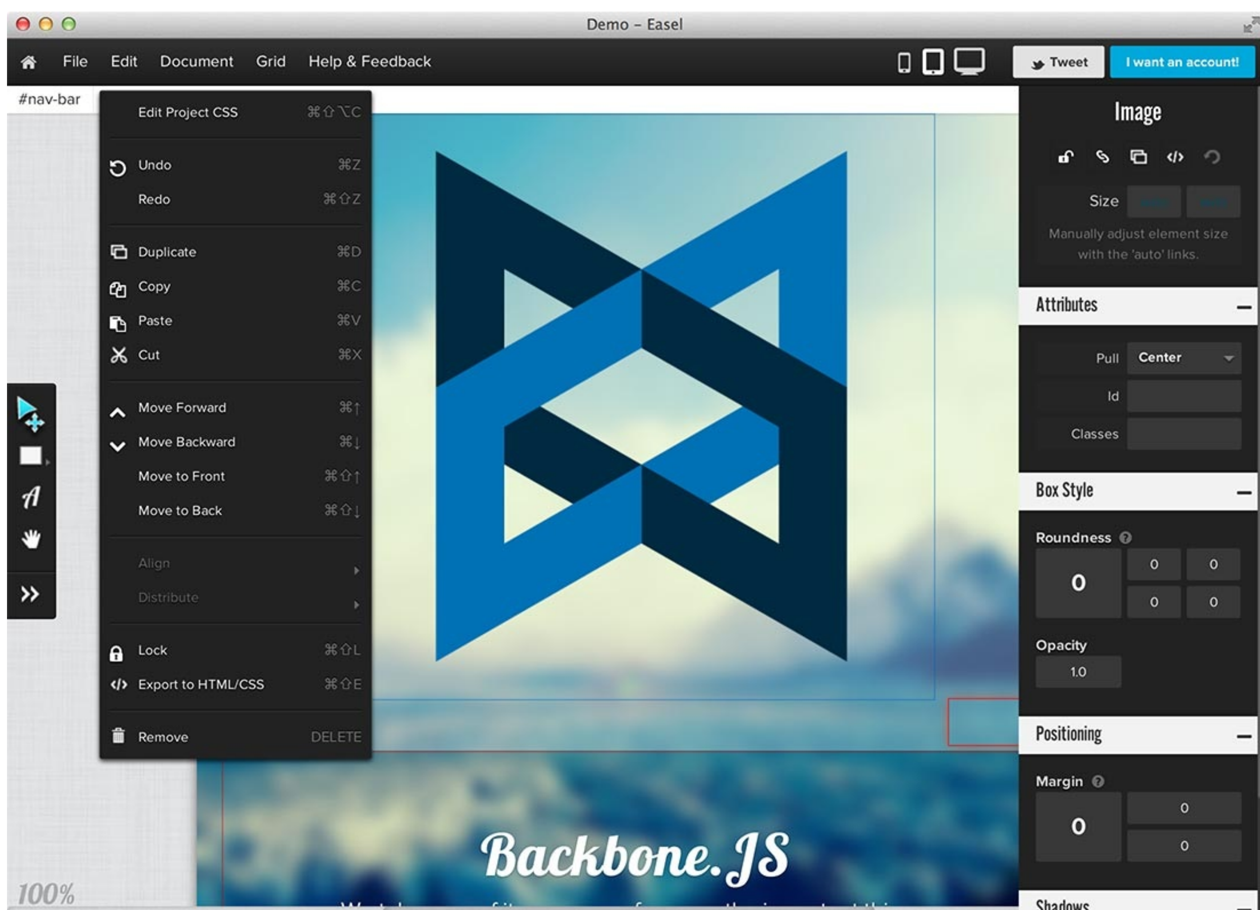



## SeatGeek

SeatGeek's stadium ticket maps were originally developed with Prototype.js. Moving to Backbone.js and jQuery helped organize a lot of the UI code, and the increased structure has made adding features a lot easier. SeatGeek is also in the process of building a mobile interface that will be Backbone.js from top to bottom.

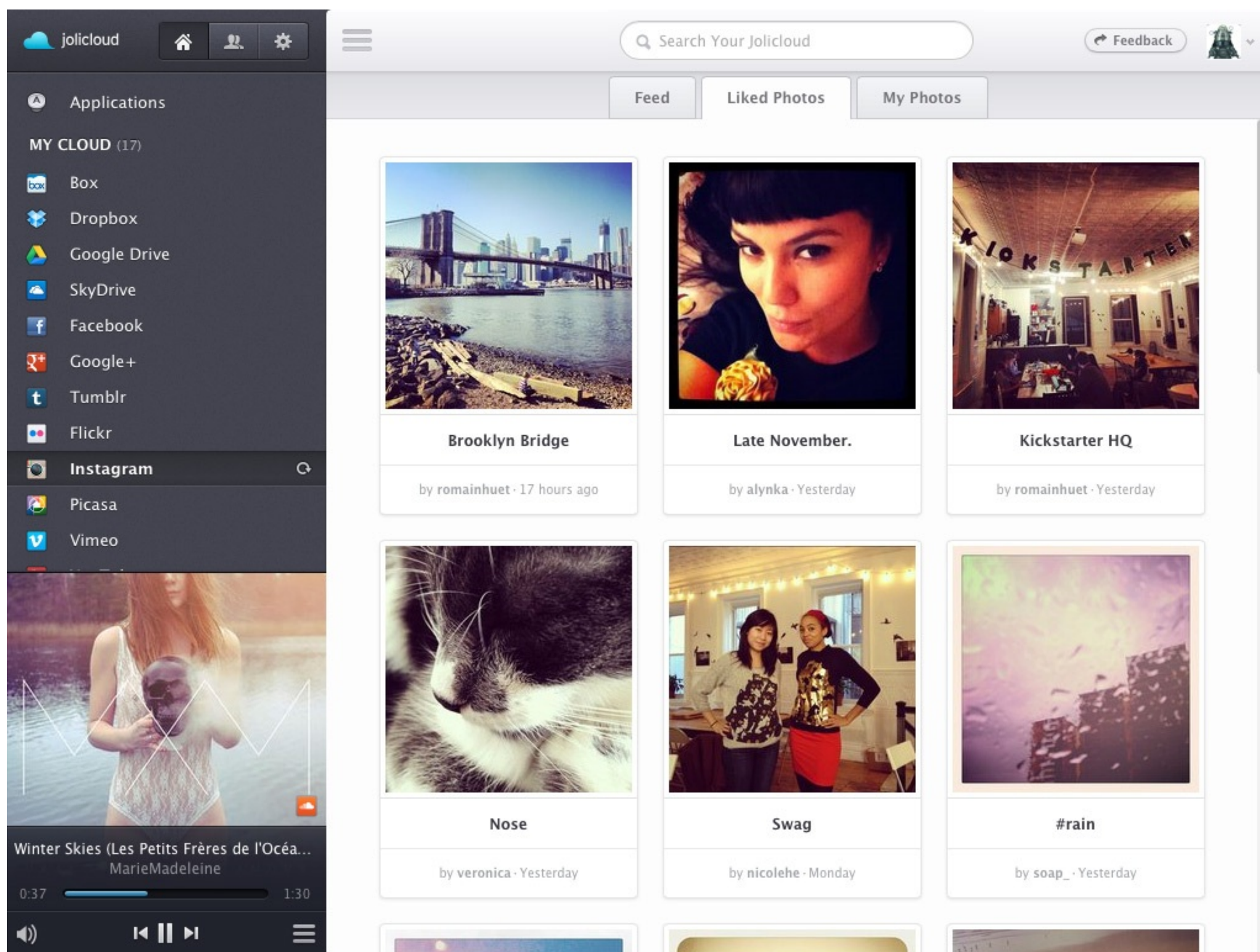
## Easel

Easel is an in-browser, high fidelity web design tool that integrates with your design and development process. The Easel team uses CoffeeScript, Underscore.js and Backbone.js for their rich visual editor as well as other management functions throughout the site. The structure of Backbone allowed the team to break the complex problem of building a visual editor into manageable components and still move quickly.



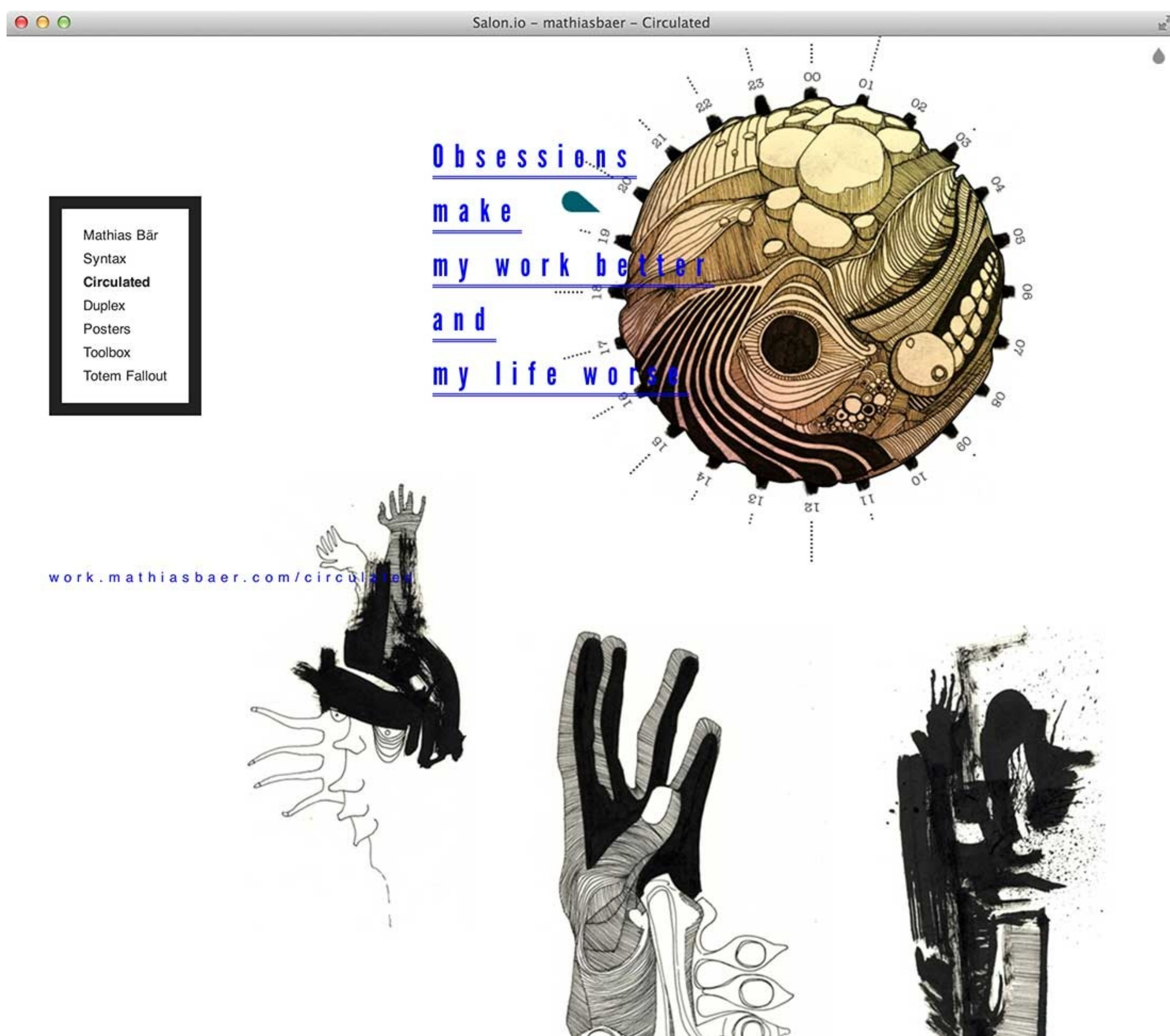

## Jolicloud

Jolicloud is an open and independent platform and operating system that provides music playback, video streaming, photo browsing and document editing — transforming low cost computers into beautiful cloud devices. The new Jolicloud HTML5 app was built from the ground up using Backbone and talks to the Jolicloud Platform, which is based on Node.js. Jolicloud works offline using the HTML5 AppCache, extends Backbone.sync to store data in IndexedDB or localStorage, and communicates with the Joli OS via WebSockets.




## Salon.io

Salon.io provides a space where photographers, artists and designers freely arrange their visual art on virtual walls. Salon.io runs on Rails, but does not use much of the traditional stack, as the entire frontend is designed as a single page web app, using Backbone.js, Brunch and CoffeeScript.

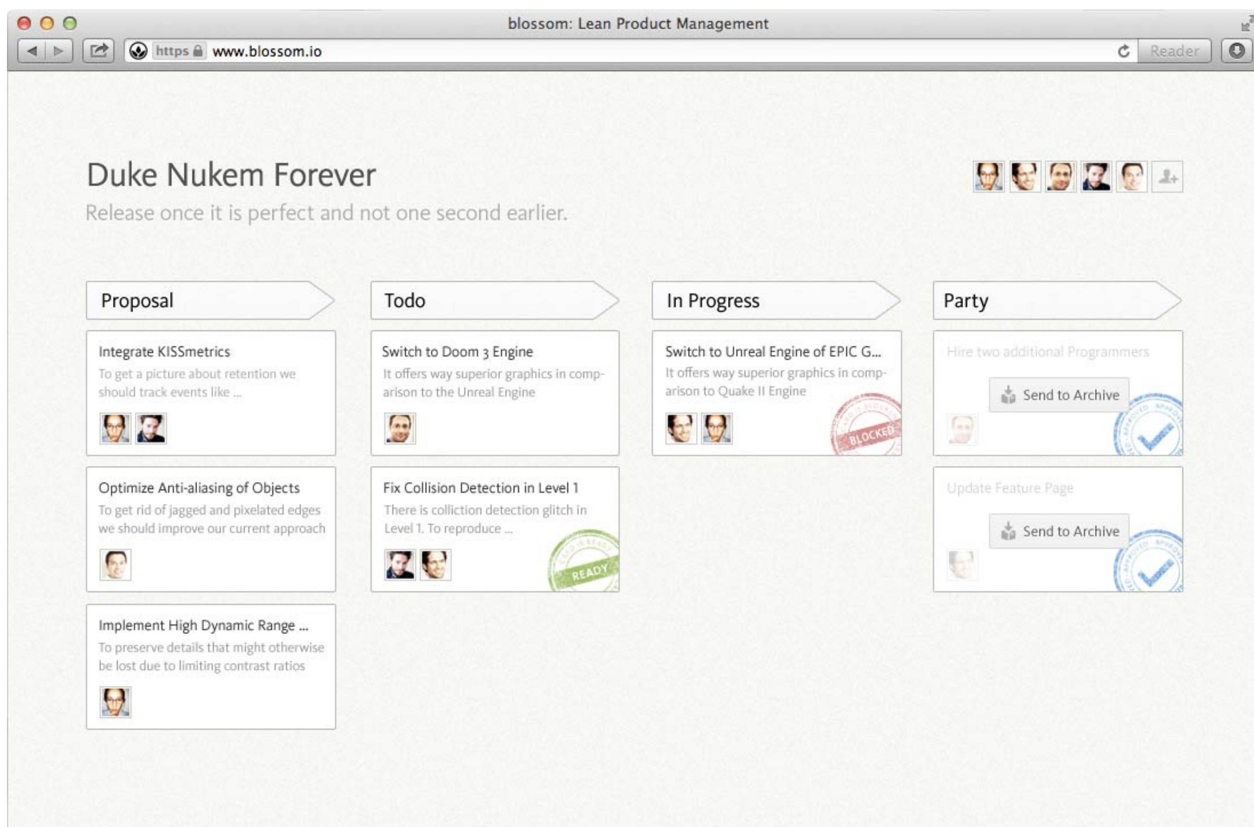



## TileMill

Our fellow Knight Foundation News Challenge winners, MapBox, created an open-source map design studio with Backbone.js: TileMill. TileMill lets you manage map layers based on shapefiles and rasters, and edit their appearance directly in the browser with the Carto styling language. Note that the gorgeous MapBox homepage is also a Backbone.js app.

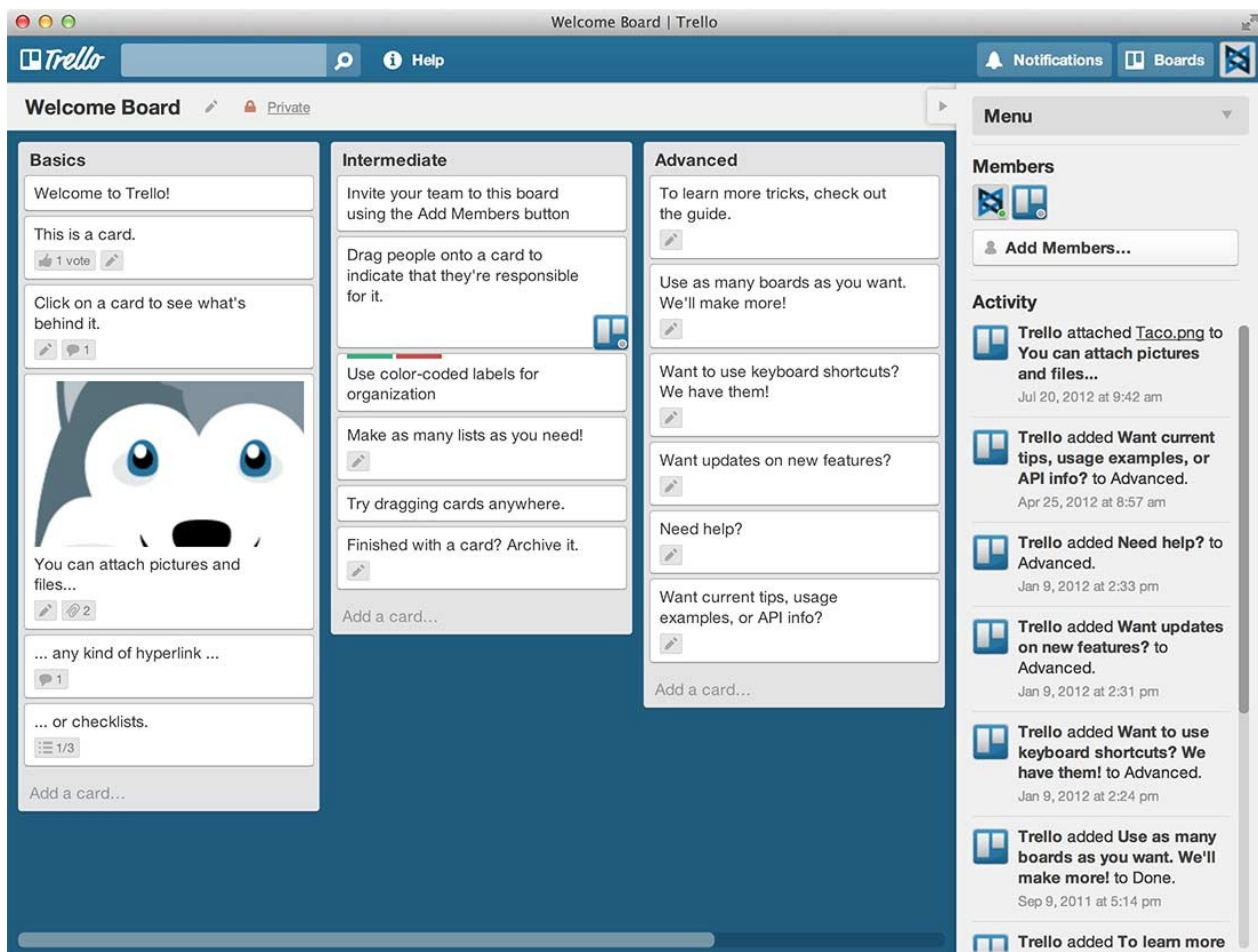
## Blossom

Blossom is a lightweight project management tool for lean teams. Backbone.js is heavily used in combination with CoffeeScript to provide a smooth interaction experience. The app is packaged with Brunch. The RESTful backend is built with Flask on Google App Engine.




## Trello

Trello is a collaboration tool that organizes your projects into boards. A Trello board holds many lists of cards, which can contain checklists, files and conversations, and may be voted on and organized with labels. Updates on the board happen in real time. The site was built ground up using Backbone.js for all the models, views, and routes.




## Tzigla

Cristi Balan and Irina Dumitrascu created Tzigla, a collaborative drawing application where artists make tiles that connect to each other to create surreal drawings. Backbone models help organize the code, routers provide bookmarkable deep links, and the views are rendered with haml.js and Zepto. Tzigla is written in Ruby (Rails) on the backend, and CoffeeScript on the frontend, with Jammit prepackaging the static assets.

# Change Log

**1.1.2** — *Feb. 20, 2014* — Diff — Docs

- Backbone no longer tries to require jQuery in Node/CommonJS environments, for better compatibility with folks using Browserify. If you'd like to have Backbone use jQuery from Node, assign it like so: `Backbone.$ = require('jquery');`
- Bugfix for route parameters with newlines in them.

**1.1.1** — *Feb. 13, 2014* — Diff — Docs

- Backbone now registers itself for AMD (Require.js), Bower and Component, as well as being a CommonJS module and a regular (Java)Script. Whew.
- Added an `execute` hook to the Router, which allows you to hook in and custom-parse route arguments, like query strings, for example.
- Performance fine-tuning for Backbone Events.
- Better matching for Unicode in routes, in old browsers.
- Backbone Routers now handle query params in route fragments, passing them into the handler as the last argument. Routes specified as strings should no longer include the query string ( `'foo?:query'` should be `'foo'` ).

**1.1.0** — *Oct. 10, 2013* — Diff — Docs

- Made the return values of Collection's `set` , `add` , `remove` , and `reset` more useful. Instead of returning `this` , they now return the changed (added, removed or updated) model or list of models.
- Backbone Views no longer automatically attach options passed to the constructor as `this.options` and Backbone Models no longer attach `url` and `urlRoot` options, but you can do it yourself if you prefer.
- All `"invalid"` events now pass consistent arguments. First the model in question, then the error object, then options.
- You are no longer permitted to change the **id** of your model during `parse` . Use `idAttribute` instead.
- On the other hand, `parse` is now an excellent place to extract and vivify incoming nested JSON into associated submodels.
- Many tweaks, optimizations and bugfixes relating to Backbone 1.0, including URL overrides, mutation of options, bulk ordering, trailing slashes, edge-case listener leaks, nested model parsing...

**1.0.0** — *March 20, 2013* — Diff — Docs

- Renamed Collection's "update" to set, for parallelism with the similar `model.set()` , and

contrast with reset. It's now the default updating mechanism after a fetch. If you'd like to continue using "reset", pass `{reset: true}`.
- Your route handlers will now receive their URL parameters pre-decoded.
- Added listenToOnce as the analogue of once.
- Added the findWhere method to Collections, similar to where.
- Added the `keys`, `values`, `pairs`, `invert`, `pick`, and `omit` Underscore.js methods to Backbone Models.
- The routes in a Router's route map may now be function literals, instead of references to methods, if you like.
- `url` and `urlRoot` properties may now be passed as options when instantiating a new Model.

**0.9.10** — *Jan. 15, 2013* — Diff — Docs

- A `"route"` event is triggered on the router in addition to being fired on `Backbone.history`.
- Model validation is now only enforced by default in `Model#save` and no longer enforced by default upon construction or in `Model#set`, unless the `{validate:true}` option is passed.
- `View#make` has been removed. You'll need to use `$` directly to construct DOM elements now.
- Passing `{silent:true}` on change will no longer delay individual `"change:attr"` events, instead they are silenced entirely.
- The `Model#change` method has been removed, as delayed attribute changes are no longer available.
- Bug fix on `change` where attribute comparison uses `!==` instead of `_.isEqual`.
- Bug fix where an empty response from the server on save would not call the success function.
- `parse` now receives `options` as its second argument.
- Model validation now fires `invalid` event instead of `error`.

**0.9.9** — *Dec. 13, 2012* — Diff — Docs

- Added listenTo and stopListening to Events. They can be used as inversion-of-control flavors of `on` and `off`, for convenient unbinding of all events an object is currently listening to. `view.remove()` automatically calls `view.stopListening()`.
- When using `add` on a collection, passing `{merge: true}` will now cause duplicate models to have their attributes merged in to the existing models, instead of being ignored.
- Added update (which is also available as an option to `fetch`) for "smart" updating of sets of models.
- HTTP `PATCH` support in save by passing `{patch: true}`.

- The `Backbone` object now extends `Events` so that you can use it as a global event bus, if you like.
- Added a `"request"` event to Backbone.sync, which triggers whenever a request begins to be made to the server. The natural complement to the `"sync"` event.
- Router URLs now support optional parts via parentheses, without having to use a regex.
- Backbone events now supports `once`, similar to Node's `once`, or jQuery's `one`.
- Backbone events now support jQuery-style event maps `obj.on({click: action})`.
- While listening to a `reset` event, the list of previous models is now available in `options.previousModels`, for convenience.
- Validation now occurs even during "silent" changes. This change means that the `isValid` method has been removed. Failed validations also trigger an error, even if an error callback is specified in the options.
- Consolidated `"sync"` and `"error"` events within Backbone.sync. They are now triggered regardless of the existence of `success` or `error` callbacks.
- For mixed-mode APIs, `Backbone.sync` now accepts `emulateHTTP` and `emulateJSON` as inline options.
- Collections now also proxy Underscore method name aliases (collect, inject, foldl, foldr, head, tail, take, and so on...)
- Removed `getByCid` from Collections. `collection.get` now supports lookup by both `id` and `cid`.
- After fetching a model or a collection, *all* defined `parse` functions will now be run. So fetching a collection and getting back new models could cause both the collection to parse the list, and then each model to be parsed in turn, if you have both functions defined.
- Bugfix for normalizing leading and trailing slashes in the Router definitions. Their presence (or absence) should not affect behavior.
- When declaring a View, `options`, `el`, `tagName`, `id` and `className` may now be defined as functions, if you want their values to be determined at runtime.
- Added a `Backbone.ajax` hook for more convenient overriding of the default use of `$.ajax`. If AJAX is too passé, set it to your preferred method for server communication.
- `Collection#sort` now triggers a `sort` event, instead of a `reset` event.
- Calling `destroy` on a Model will now return `false` if the model `isNew`.
- To set what library Backbone uses for DOM manipulation and Ajax calls, use `Backbone.$ = ...` instead of `setDomLibrary`.
- Removed the `Backbone.wrapError` helper method. Overriding `sync` should work better for those particular use cases.
- To improve the performance of `add`, `options.index` will no longer be set in the `add` event callback. `collection.indexOf(model)` can be used to retrieve the index of a model as necessary.
- For semantic and cross browser reasons, routes will now ignore search parameters.

Routes like `search?query=…&page=3` should become `search/…/3`.
- `Model#set` no longer accepts another model as an argument. This leads to subtle problems and is easily replaced with `model.set(other.attributes)`.

**0.9.2** — *March 21, 2012* — Diff — Docs

- Instead of throwing an error when adding duplicate models to a collection, Backbone will now silently skip them instead.
- Added push, pop, unshift, and shift to collections.
- A model's changed hash is now exposed for easy reading of the changed attribute delta, since the model's last `"change"` event.
- Added where to collections for simple filtering.
- You can now use a single off call to remove all callbacks bound to a specific object.
- Bug fixes for nested individual change events, some of which may be "silent".
- Bug fixes for URL encoding in `location.hash` fragments.
- Bug fix for client-side validation in advance of a `save` call with `{wait: true}`.
- Updated / refreshed the example Todo List app.

**0.9.1** — *Feb. 2, 2012* — Diff — Docs

- Reverted to 0.5.3-esque behavior for validating models. Silent changes no longer trigger validation (making it easier to work with forms). Added an `isValid` function that you can use to check if a model is currently in a valid state.
- If you have multiple versions of jQuery on the page, you can now tell Backbone which one to use with `Backbone.setDomLibrary`.
- Fixes regressions in **0.9.0** for routing with "root", saving with both "wait" and "validate", and the order of nested "change" events.

**0.9.0** — *Jan. 30, 2012* — Diff — Docs

- Creating and destroying models with `create` and `destroy` are now optimistic by default. Pass `{wait: true}` as an option if you'd like them to wait for a successful server response to proceed.
- Two new properties on views: `$el` — a cached jQuery (or Zepto) reference to the view's element, and `setElement`, which should be used instead of manually setting a view's `el`. It will both set `view.el` and `view.$el` correctly, as well as re-delegating events on the new DOM element.
- You can now bind and trigger multiple spaced-delimited events at once. For example: `model.on("change:name change:age", ...)`
- When you don't know the key in advance, you may now call `model.set(key, value)` as well as `save`.
- Multiple models with the same `id` are no longer allowed in a single collection.
- Added a `"sync"` event, which triggers whenever a model's state has been successfully

synced with the server (create, save, destroy).

- `bind` and `unbind` have been renamed to `on` and `off` for clarity, following jQuery's lead. The old names are also still supported.
- A Backbone collection's `comparator` function may now behave either like a sortBy (pass a function that takes a single argument), or like a sort (pass a comparator function that expects two arguments). The comparator function is also now bound by default to the collection — so you can refer to `this` within it.
- A view's `events` hash may now also contain direct function values as well as the string names of existing view methods.
- Validation has gotten an overhaul — a model's `validate` function will now be run even for silent changes, and you can no longer create a model in an initially invalid state.
- Added `shuffle` and `initial` to collections, proxied from Underscore.
- `Model#urlRoot` may now be defined as a function as well as a value.
- `View#attributes` may now be defined as a function as well as a value.
- Calling `fetch` on a collection will now cause all fetched JSON to be run through the collection's model's `parse` function, if one is defined.
- You may now tell a router to `navigate(fragment, {replace: true})`, which will either use `history.replaceState` or `location.hash.replace`, in order to change the URL without adding a history entry.
- Within a collection's `add` and `remove` events, the index of the model being added or removed is now available as `options.index`.
- Added an `undelegateEvents` to views, allowing you to manually remove all configured event delegations.
- Although you shouldn't be writing your routes with them in any case — leading slashes ( `/` ) are now stripped from routes.
- Calling `clone` on a model now only passes the attributes for duplication, not a reference to the model itself.
- Calling `clear` on a model now removes the `id` attribute.

**0.5.3** — *August 9, 2011* — Diff — Docs A View's `events` property may now be defined as a function, as well as an object literal, making it easier to programmatically define and inherit events. `groupBy` is now proxied from Underscore as a method on Collections. If the server has already rendered everything on page load, pass `Backbone.history.start({silent: true})` to prevent the initial route from triggering. Bugfix for pushState with encoded URLs.

**0.5.2** — *July 26, 2011* — Diff — Docs The `bind` function, can now take an optional third argument, to specify the `this` of the callback function. Multiple models with the same `id` are now allowed in a collection. Fixed a bug where calling `.fetch(jQueryOptions)` could cause an incorrect URL to be serialized. Fixed a brief extra route fire before redirect, when degrading from `pushState`.

**0.5.1** — *July 5, 2011* — Diff — Docs Cleanups from the 0.5.0 release, to wit: improved transparent upgrades from hash-based URLs to pushState, and vice-versa. Fixed inconsistency with non-modified attributes being passed to `Model#initialize`. Reverted a **0.5.0** change that would strip leading hashbangs from routes. Added `contains` as an alias for `includes`.

**0.5.0** — *July 1, 2011* — Diff — Docs A large number of tiny tweaks and micro bugfixes, best viewed by looking at the commit diff. HTML5 `pushState` support, enabled by opting-in with: `Backbone.history.start({pushState: true})`. `Controller` was renamed to `Router`, for clarity. `Collection#refresh` was renamed to `Collection#reset` to emphasize its ability to both reset the collection with new models, as well as empty out the collection when used with no parameters. `saveLocation` was replaced with `navigate`. RESTful persistence methods (save, fetch, etc.) now return the jQuery deferred object for further success/error chaining and general convenience. Improved XSS escaping for `Model#escape`. Added a `urlRoot` option to allow specifying RESTful urls without the use of a collection. An error is thrown if `Backbone.history.start` is called multiple times. `Collection#create` now validates before initializing the new model. `view.el` can now be a jQuery string lookup. Backbone Views can now also take an `attributes` parameter. `Model#defaults` can now be a function as well as a literal attributes object.

**0.3.3** — *Dec 1, 2010* — Diff — Docs Backbone.js now supports Zepto, alongside jQuery, as a framework for DOM manipulation and Ajax support. Implemented Model#escape, to efficiently handle attributes intended for HTML interpolation. When trying to persist a model, failed requests will now trigger an `"error"` event. The ubiquitous `options` argument is now passed as the final argument to all `"change"` events.

**0.3.2** — *Nov 23, 2010* — Diff — Docs Bugfix for IE7 + iframe-based "hashchange" events. `sync` may now be overridden on a per-model, or per-collection basis. Fixed recursion error when calling `save` with no changed attributes, within a `"change"` event.

**0.3.1** — *Nov 15, 2010* — Diff — Docs All `"add"` and `"remove"` events are now sent through the model, so that views can listen for them without having to know about the collection. Added a `remove` method to Backbone.View. `toJSON` is no longer called at all for `'read'` and `'delete'` requests. Backbone routes are now able to load empty URL fragments.

**0.3.0** — *Nov 9, 2010* — Diff — Docs Backbone now has Controllers and History, for doing client-side routing based on URL fragments. Added `emulateHTTP` to provide support for legacy servers that don't do `PUT` and `DELETE`. Added `emulateJSON` for servers that can't accept `application/json` encoded requests. Added Model#clear, which removes all attributes from a model. All Backbone classes may now be seamlessly inherited by CoffeeScript classes.

**0.2.0** — *Oct 25, 2010* — Diff — Docs Instead of requiring server responses to be namespaced under a `model` key, now you can define your own parse method to convert responses into attributes for Models and Collections. The old `handleEvents` function is now named delegateEvents, and is automatically called as part of the View's constructor. Added a toJSON function to Collections. Added Underscore's chain to Collections.

**0.1.2** — *Oct 19, 2010* — Diff — Docs Added a Model#fetch method for refreshing the attributes of single model from the server. An `error` callback may now be passed to `set` and `save` as an option, which will be invoked if validation fails, overriding the `"error"` event. You can now tell backbone to use the `_method` hack instead of HTTP methods by setting `Backbone.emulateHTTP = true`. Existing Model and Collection data is no longer sent up unnecessarily with `GET` and `DELETE` requests. Added a `rake lint` task. Backbone is now published as an NPM module.

**0.1.1** — *Oct 14, 2010* — Diff — Docs Added a convention for `initialize` functions to be called upon instance construction, if defined. Documentation tweaks.

**0.1.0** — *Oct 13, 2010* — Docs Initial Backbone release.